LATVIJAS UNIVERSITĀTE
DATORIKAS FAKULTĀTE

UZ ATTĒLIEM BALSTĪTA RENDERĒŠANA

BAKALAURA DARBS

Autors: Sergejs Stavro
Stud. apl. ss07069
Darba vadītājs: Dr.dat., docents, Kārlis Freivalds

RĪGA 2011

Anotācija

Šajā darba tiek aplūkotas izplatītākas uz attēliem balstītas renderēšana metodes. Ka arī izmantojot skatu metamorfēšanu, tiek izveidota programma, kas atrod starpattēlu diviem kadriem, kas tika ņemti dažādos laika momentos.

Atslēgvārdi

Uz attēliem balstīta renderēšana, skatu metamorfēšana.

Abstract

This work covers the most widespread methods of image-based rendering. Also with the use of view morphing a piece of software is built that finds an intermediate image based on two input images that were made in different time moments.

Keywords

Image-based rendering, view morphing

Satura rādītājs

Apzīmējumu saraksts.

Image-based rendering (IBR) - uz attēliem balstīta renderēšana.

Plenoptic modeling – plenoptiskā modelēšana.

Light field – gaismas lauks.

Lumigraph – lumigrafs.

Warping function – deformācijas funkcija.

View interpolation – skatu interpolācija.

View morphing – skatu metamorfēšana.

Transfer methods – pārsūtīšanas metodes.

3D warping – 3D deformācijas.

Layered depth images (LDI) – slāņoti dziļuma attēli.

View-dependent texture maps – no skata atkarīgas faktūru kartes.

Computer-aided design (CAD) – projektēšana ar datora palīdzību.

Splatting – splatēšana.

Billboard – stends.

Omnidirectional camera – izkliedēta kamera.

Foldover – pārklāšanas.

Ievads.

Problēma, kuru es centīšos risināt šinī darbā ir sekojoša: Ir doti divi ceļa satiksmes attēli, kas tika nofilmēti dažādos laika momentos, nepieciešams rekonstruēt šo divu attēlu starpattēlu, t.i. bildi, kas attēlotu apkārtējo vidi laika momentā starp laika momentiem, kuros tika veidoti ieejas attēli.

Galvenais darba mērķis ir atrisināt uzstādītu problēmu un apskatīt uz attēliem balstītas renderēšanas metodes.

Parasti 3D sistēmas ir dota ģeometriskā informācija, materiāli un gaismu kopas, kas apraksta scēnu. Balstoties uz šo informāciju sistēma var no jebkura telpas punkta ģenerēt jaunu attēlu. Uz attēliem balstītam renderēšanas sistēmām ir cits piegājiens, tās ģenerē jaunus skatus bastoties uz iepriekš definētas attēlu kopas. Šim piegājienam ir vairākas priekšrocības:

- Attēlošanas algoritmi var strādāt reālajā laikā.
- Scēnas sarežģītība nespēlē nekādu lomu.
- Iepriekš definēta attēlu kopa var būt gan no reālas, gan no virtuālas vides.

Šī darba struktūra ir sekojoša: Pirmā daļa ir teorētiskā, kur tiks apskatītas vairākas esošas uz attēliem balstītas renderēšanas metodes. Otrajā daļā tiks aprakstīts praktiskais darbs, ko es paveicu, lai risinātu doto problēmu. Trešā daļa satur šī darba rezultātu aprakstu. Ceturtā daļā apkopo secinājumus, kas radās padarīta darba rezultātā.

1 Teorētiskā daļa

1.1 Renderēšana bez ģeometrijas

Šajā nodaļā tiks aprakstītas metodes, kas ir pielietojamas, kad nav zināma ainas/skata ģeometrija. Šis metodes vairumā balstās uz daudziem ieejas attēliem un plenoptisko funkciju.

1.1.1 Plenoptiskā modelēšana

Adelsons and Bergens (3) uzskata, ka viens no agras redzes uzdevumiem ir kompakta un noderīga plenoptiskās funkcijas lokālo īpašību apraksta ieguve. (1)

7D plenoptiskā funkcija tiek definēta, ka gaismas staru, kas iziet cauri katram telpas punktam(*Vx, Vy, Vz*), visos iespējamos leņķos (θ, ϕ), katram viļņa garumam λ, visos laika momentos *t*, intensitāte. (3)

$$\mathrm{P} = \mathrm{P}\left(\theta, \phi, \lambda, t, V_x, V_y, V_z\right)$$

Gadījumā, kad mēs aprakstam statisku melnbaltu vidi, modelēšanu var veikt ar 5D plenoptisko funkciju. Izņemot no funkcijas divus mainīgus, laiku un viļņa garumu McMillans and Bišofs ieviesa plenoptiskas modelēšanas ar pilnīgu 5D plenoptisko funkciju jēdzienu. 5D plenoptiskā funkcija izskatās sekojoši (1)

$$\mathrm{P} = \mathrm{P}\left(\theta, \phi, V_x, V_y, V_z\right)$$

Visvienkāršākā plenoptiskā funkcija ir sfēriska vai cilindriska 2D panorāma ar fiksētu skata punktu. Parasts taisnstūra attēls var būt uzskatīts ka plenoptiskās funkcijas nepilnīgs paraugs ar fiksētu skata punktu.

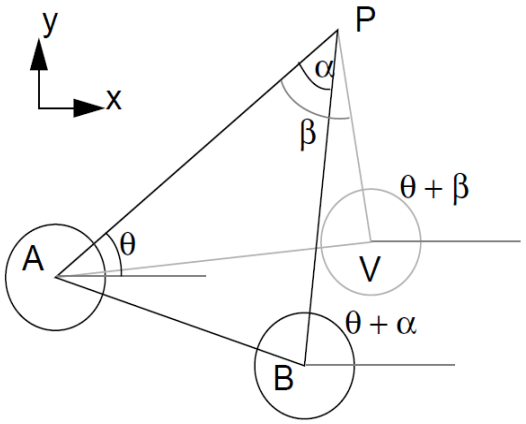
$$\mathrm{P} = \mathrm{P}(\theta, \phi)$$

Uz attēliem balstīta renderēšana var būt uzskatīta, ka metožu kopa nepārtrauktas plenoptiskās funkcijas rekonstruēšanai no diskrētiem paraugiem. Paraugu ņemšana un nepārtrauktas funkcijas rekonstruēšana no diskrētiem paraugiem ir svarīgi uz attēliem balstītas renderēšanas jautājumi. (1)

Plenoptiskajā modelēšanā, tāpat kā citās uz attēliem balstītas renderēšanas sistēmās, aina tiek aprakstīta ar atskaites attēlu virkni. Šie atskaites attēli pēc tam tiek deformēti un apvienoti, lai izveidotu ainas reprezentāciju no patvaļīga skata punkta. Deformācijas funkcija ir definēta ar attēlu plūsmas lauka informāciju, kas var būt vai nu piegādāta kā ieeja vai ir atvasināta no atskaites attēliem. (4)

Cilindriskas panorāmas, kas tika izmantotas McMillana and Bišofa darbā "Plenoptiskā modelēšana" ir plenoptiskās funkcijas divdimensionāli paraugi no diviem skata punktiem. Divi panorāmu skata virzieni katrai panorāmai panoramējas un noliecas ap tas centru. [Šo ierobežojumu var atvieglot, ja par scēnu ir zināma ģeometriskā informācija.] McMillana

darbā stereo metodes tiek pielietotas vairākām panorāmām, lai dabūtu disparitātes sadali. Šīs sadales var vēlāk būt izmantotas, lai prognozētu patvaļīgu vietu izskatu. (1) Atrodot katra pikseļa disparitāti divu cilindrisku attēlu stereo pārī, kas ir aptuveni ekvivalents ar dziļuma informācijas atrašanu, var pārvietot pikseļus attēlu ģenerēšanai no jauniem skatu punktiem.

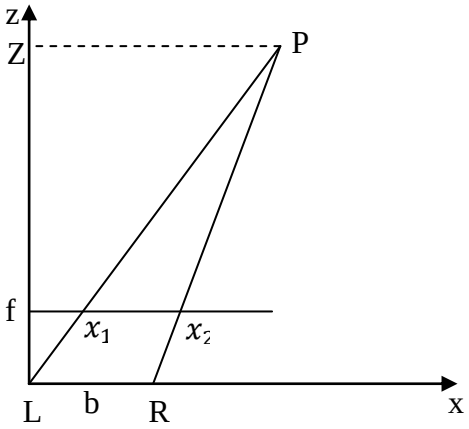


Att. 1.1.1 Jauna cilindra veidošana (4)

Paralēlam kamerām pikseļu dziļumu, balstoties uz pikseļu pozīciju atšķirībām, var atrast pēc formulas:

$$Z = (b * f)/(x_1 - x_2),$$

kur Z ir dziļums, b – attālums starp kameru centriem, f – fokuss un x1,x2 ir pikseļa koordinātes dažādos attēlos.



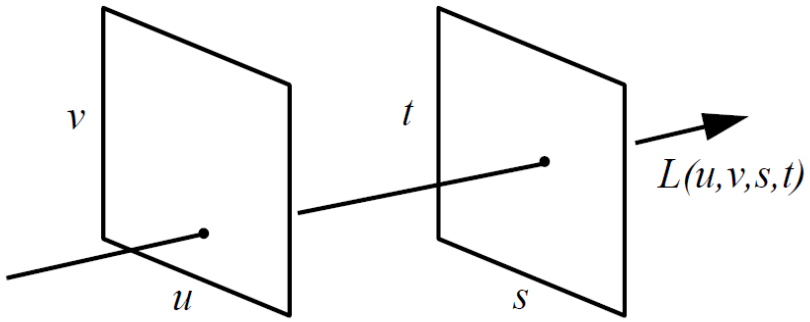
Att. 1.1.2 Punkts stereo pārī

1.1.2 Gaismas lauks un lumigrafs

Tika pamanīts, ka gadījumā, kad mēs paliekam ārpus objekta izliektas čaulas, mēs varam 5D plenoptisko funkciju novienkāršot līdz 4D plenoptiskai funkcijai,

$$P = P(u, v, s, t)$$

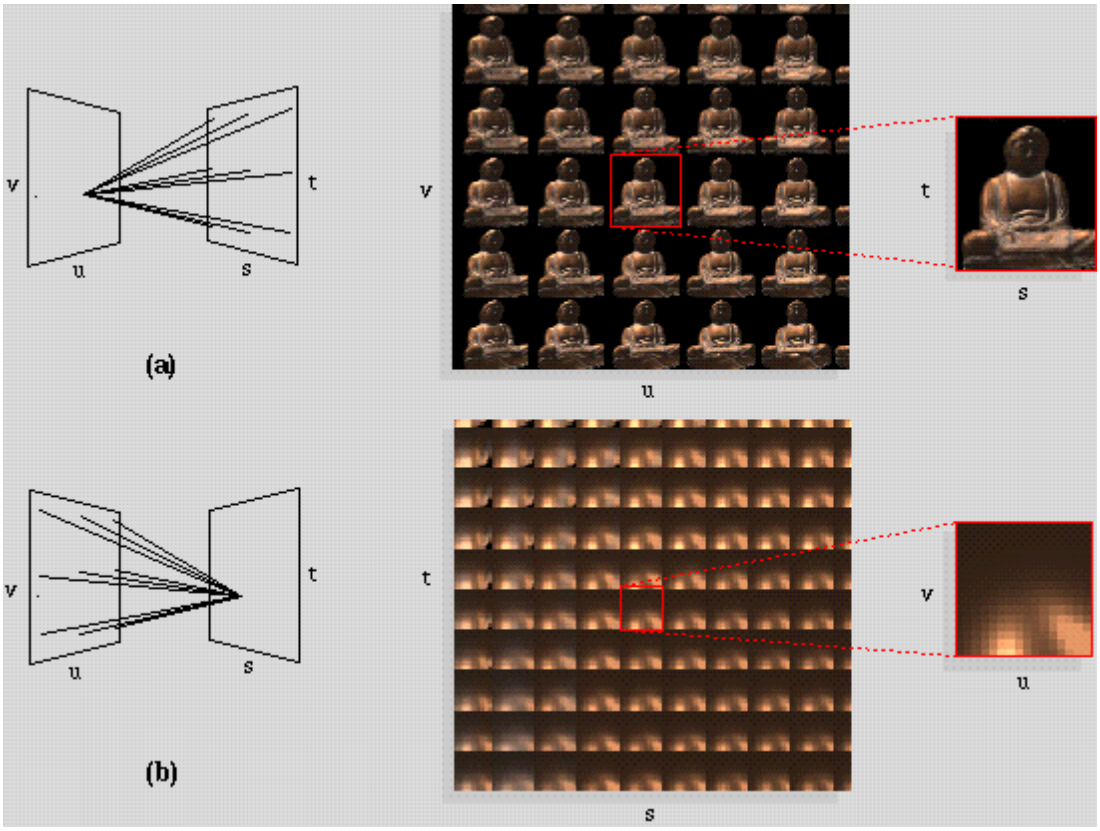
kur (u, v) un (s, t) apraksta divas plaknes, kas ir paralēlas, objektu ierobežojošai, kastei, kas ir redzams attēlā (Att. 1.1.3). (5)



Att. 1.1.3 Gaismas plākšņu reprezentācija (5)

Plakne (u, v) ir kameras plakne, kur atrodas paraugu ņemšanas kameras. Attēls (Att. 1.1.4 a) parāda gaismas lauka vizualizāciju no kameras plaknes. No punkta, kas atbilst paraugu ņemšanas kameras pozīcijai, skats ir oriģinālas. (1)

Levoja and Hanrahana gaismas lauka sistēmai (s, t) ir fokāla plakne, tiek pieņemts, ka tur atrodas scēna. Gaismas lauka vizualizācija no fokālas plaknes ir uzradīta attēlā (Att. 1.1.4 b). (1)



Att. 1.1.4 Gaismas lauka vizualizācijas (5)

Pieņemot, ka scēnas virsma ir aptuveni fokālajā plaknē, visi stari, kas iziet cauri punktam fokālajā plaknē ir vienas virsmas izskata paraugi no dažādiem skatiem. Tas līdzās divvirzienu atstarošanas sadalījuma funkcijas atrašanai fiksētam apgaismojumam. Stari tiek interpolēti balstoties uz pieņēmuma, ka scēnas virsma ir tuvu fokālai plaknei. Objektu virsmas, kas atrodas tālu no fokālas plaknes interpolētos skatos izskatīsies izplūduši. Lumigrafs interpolācijai izmanto aptuvenu 3D objekta virsmu, kas mazina izpludināšanas problēmu. Svarīgi piezīmēt, ka lumigrafam (u, v) ir fokāla plakne un (s, t) ir kameras plakne. (1)

Principā plaknēm (u, v) un (s, t) nav obligāti jābūt paralēlam. Vēl tiek pieņēmis, ka gaismas stara stiprums nemainās tā ceļa garumā. Lai dabūtu pilnu plenoptisku funkciju ierobežojošai kastei, ir nepieciešami seši divu plakņu pāri. (1) Katrā telpas punktā var noteikt spožumu katram staram jebkurā virzienā, trasējot šo staru atpakaļ caur tukšu telpu līdz kuba virsmai.

Gaismas lauka renderēšanas un lumigrafa darbības principi ir ļoti līdzīgi, izņēmums ir tas ka lumigrafam papildus ir objektu ģeometrija labākai kompresijai un izskata pareģošanai. Tādēļ lumigrafs principā pieder pie metodēm, kas izmanto izteiktu ģeometriju, bet metožu līdzības dēļ tiek aprakstīts šeit. (1)

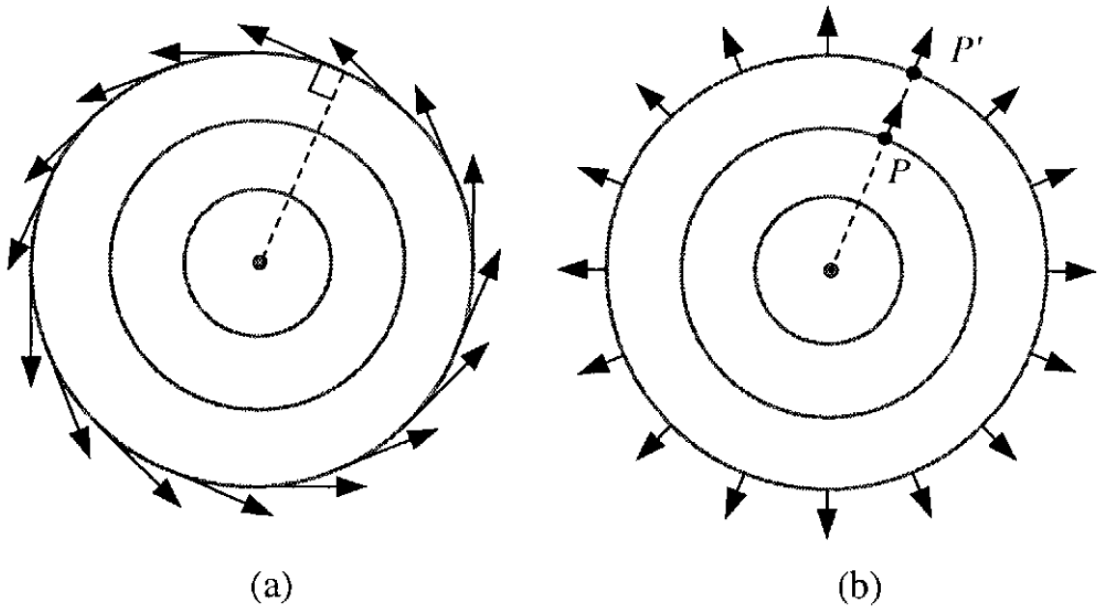
Gaismas lauka sistēma izmanto notveršanas stendu, vienāda tipa paraugu attēlu veidošanai. Lai samazinātu kropļošanas efektu, gaismas lauks tiek filtrēts pirms renderēšanas. Vektoru kvantēšanas shēma tiek izmantota izmantojamu datu apjoma samazināšanai. Savukārt lumigrafs var būt konstruēts no attēlu kopas, kas bija nofilmēti no patvaļīgiem skatu punktiem. Tāpēc ir nepieciešams process, kas no jauna veidotu visus paraugus, izmantojot hierarhisko interpolācijas shēmu, tā, lai tie atbilstu regulāram tiklam. Paraugu blīvums var būt samazināts pateicoties ģeometriskas informācijas izmantošanai. (1)

Divu plakņu parametru iestatīšana $P(u, v, s, t)$ ir tikai viena no iespējamam gaismas laukiem, eksistē arī sfēriskie un vairāki citi gaismas lauki. (1)

1.1.3 Koncentriskas mozaīkas

Jo vairāk ierobežojumu mēs ieviešam, jo vienkāršāka paliek plenoptiskā funkcija. Koncentriskas mozaīkas ir plenoptiskās funkcijas 3D parametru iestatījums, kurā kameras kustība tiek ierobežota ar plaknes koncentriskiem riņķiem.

Ierobežojot kameras kustību līdz plaknes koncentriskiem riņķiem, tiek panākts tas, ka mozaīkas var tikt saliktas no vairākiem attēliem, kas tika ņemti dažādās vietās katrā riņķī. Koncentriskas mozaīkas var tikt ņemtas dažādos veidos, divi piemēri varētu būt, kad stari ir organizēti tangenciālā virzienā un, kad stari it organizēti normāles virziena, attiecībā pret riņķi. (1)



Att. 1.1.5 Koncentrisku mozaīku tipi (1)

Koncentriskas mozaīkas definē 3D plenoptisko funkciju, jo paraugi tiek ņemti dabiskajā veidā izmantojot trīs parametrus: rotācijas leņķis, rādiuss un vertikāls paaugstinājums. Skaidri saprotams, ka pikseļi tiek kartēti viens pret vienu koncentriskajā mozaīkā un atbilstošajā scēnas punktā. (1)

Jauni skati tiek renderēti kombinējot atbilstošus notvertus starus renderēšanas laikā. Kaut gan renderētos attēlos eksistē vertikāla dispozīcija, to var mazināt izmantojot dziļuma korekciju. Koncentriskām mozaīkām ir laba telpas un skaitļošanas efektivitāte. Samērā ar gaismas lauku vai lumigrafu, koncentriskajam mozaīkām ir mazāks faila izmērs, jo tiek konstruēta tikai 3D funkcija. (1)

Koncentrisko mozaīku notveršana ir gandrīz tik pat vienkārša, ka tradicionālu panorāmu notveršana, izņemot to, ka koncentriskajām mozaīkām ir nepieciešams vairāk attēlu. Vienkārši rotējot kameru, kas nav pozicionēta speciālas instalācijas centra, var konstruēt reālas scēnas koncentrisko mozaīku 10 minūšu laikā. Tā pat, ka panorāmām, koncentriskajām mozaīkām nav nepieciešams sarežģīts modelēšanas process ģeometrisku datu un fotometrisku scēnu modeļu dabūšanai. Tomēr, koncentriskas mozaīkas sniedz lietotājam daudz bagātāku pieredzi, ļaujot brīvi pārvietoties pa riņķi un noverot ievērojamas paralakses un apgaismojuma izmaiņas. Koncentrisko mozaīku veidošanas vieglums padara tās par ļoti pievilcīgu metodi daudzām virtuālas realitātes programmām. (1)

Tika parādīts, ka jauni skati notveršanas riņķa iekšpusē var būt renderēti izmantojot koncentriskas mozaīkas bez jebkādas informācijas par scēnas dziļumu. No koncentriskas mozaīkas ar blīvu paraugu kopu jauns skats var tikt renderēts, izmantojot lineāro interpolāciju tuviem stariem no kaimiņu koncentriskajam mozaīkām. Papildus tiek pieņemts, ka dziļums ir konstants, tas ļauj atrast labākus “tuvus” starus optimālai

renderēšanas kvalitātei. Neskatoties uz neizbēgamiem vertikāliem kropļojumiem, koncentriskas mozaīkas ir ļoti noderīgas virtuālas vides izpētīšanai. (1)

Att. 1.1.6 Saliktas kopā koncentriskas mozaīkas (2)

1.1.4 Attēlu mozaīku veidošana

Pilnīga plenoptiska funkcija fiksētā skatu punktā var tikt konstruēta no nepilnīgiem paraugiem. Panorāmas mozaīka tiek veidota no vairākiem regulāriem attēliem. Piemēram, gadījumā kad kameras fokusa attālums ir zināms un fiksēts, katru attēlu var projicēt uz cilindrisku karti, tad attiecība starp cilindriskiem attēliem kļūst par vienkāršu translāciju. Patvaļīgai kameras rotācijai var vispirms reģistrēt attēlu iegūstot kameras kustību, pirms konvertēšanas beidzamajā cilindriskajā/sfēriskajā kartē. (1)

Daudzas sistēmas, kas konstruē cilindriskas un sfēriskas panorāmas, saliekot vairākus attēlus kopa, jau tika uzbūvētas. Kad kameras kustība ir neliela, ir iespējams uzbūvēt nelielas svītras no reģistrētiem attēliem, piemēram šķeltus attēlus, lai veidotu lielu panorāmas mozaīku. Panorāmu notveršana ir vēl vieglāka, ja tiek izmantotas izkliedētas kameras vai zivs acs tipa objektīvs. (1)

Zeliskis and Šums prezentēja gatavu sistēmu, kas var konstruēt attēlu panorāmu mozaīkas no attēlu kopas. Viņu mozaīkas reprezentācija asociē transformācijas matricu ar katru ieejas attēlu, nevis projicē visus attēlus uz vienas virsmas, piemēram uz cilindra. Lai konstruētu pilnu skata panorāmu, rotācijas matricas reprezentācija asociē rotācijas matricu ar katru no ieejas attēliem. Uz ielāpiem balstīts līdzināšanas algoritms tika veidots, lai ātri līdzināt divus attēlus, ja ir doti kustību modeļi. Kameras fokāla attāluma novērtēšanas un rafinēšanas tehnikas arī tiek prezentētas. (1)

Att. 1.1.7 Attēlu mozaīkas (12)

Lai samazinātu uzkrāto reģistrācijas kļūdu skaitu, visai attēlu kopai tiek pielietota globāla līdzināšana, kas izmanto bloku koriģēšanu, kas rezultātā dod optimāli reģistrētu attēlu mozaīku. Lai kompensētu nelielu kustības paralaksi, ko ievieš kameras translācija, un citus kropļojumus, lokālas līdzināšanas metode deformē katru attēlu balstoties uz lokālas attēlu pāru reģistrācijas rezultātiem. Apvienojot globālo un lokālo līdzināšanu jūtami uzlabojas attēlu mozaīku kvalitāte, kas dod iespēju veidot pilna skata panorāmu mozaīkas turot kameru rokās. (1)

Vairākus attēlus var ne tikai sajaukt kopā, lai producētu plašākus redzeslaukus, bet tos var izmantot arī lai veidotu augstākas izšķirtspējas panorāmas. Eksistē arī metodes, kas apstrādā ieejas attēlus ar ekspozīcijas atšķirībām. (1)

1.1.5 Secinājumi

Metodes, kas neizmanto scēnas ģeometriju, lielā mērā paļaujas uz blīvu avota attēlu kopu, tas izmanto atbilstības starp attēliem, lai iegūtu jaunus skatus. Šo metožu trūkums ir tas, ka tās pieprasa daudz fiziskās atmiņas, bet plusi ir, ka metodes ir samēra viegli saprast un viegli realizēt. Galvenais metožu pielietošanas veids – panorāmu veidošana.

1.2 Renderēšana ar neizteiktu ģeometriju

Metodes, kas tika aprakstītas iepriekšējā nodaļā, lai veidotu virtuālus skatus, ņem paraugus tieši no avota attēliem. Relatīvas transformācijas starp kamerām vai optisko plūsmu laukiem tiek izrēķināti pārsvarā, lai stabilizētu panorāmu veidošanu. Šajā nodaļa tiks aprakstītas metodes, kas balstās uz pozicionālam atbilstībām starp nelielu attēlu skaitu, jaunu skatu renderēšanai. Šī klase ir aprakstīta ar vārdiem "neizteikta ģeometrija", jo tiek pieņemts, ka šīs klases metodēm ģeometrija nav tieši pieejama, 3D informācija tiek izrēķināta izmantojot tikai parastu projekciju aprēķinus. Dažos gadījumos, kad kameras ir vāji nokalibrētas, 3D informācija nav pieejama pat, ja ir pieejama atbilstību informācija. Jauni skati tiek izrēķināti balstoties uz tiešam pozicionālo atbilstību manipulācijām, kas pārsvara ir punkti. (1)

Šis klases metodes ir skatu interpolācija, skatu metamorfēšana, savienotu skatu interpolācija, un pārsūtīšanas metodes ar fundamentālo matricu un trifokālo tenzoru. Skatu interpolācija izmanto blīvu optisko plūsmu, lai pa tiešo ģenerētu starp-skatus. Starp-skati var arī nebūt ģeometriski korekti. Skatu metamorfēšana ir skatu interpolācijas speciālais gadījums, tas atšķiras ar to, ka skatu metamorfēšanas rezultāti ir vienmēr ģeometriski korekti. Ģeometriskais korektums tiek nodrošināts ar kameras lineāru kustību. Pārsūtīšanas metodes arī producē ģeometriski pareizus skatus, tie atšķiras ar to ka kameras skata punkts var būt patvaļīgi pozicionēts. (1)

1.2.1 Skatu interpolācija

Skatu interpolācija ļauj veidot jaunu attēlu balstoties uz saglabātiem blakus skatu punktu attēliem. Attēla ģenerēšana nav atkarīga no scēnas sarežģītības. Blakus attēlus metamorfē, lai dabūtu jaunu attēlu skatu punktam, kas atrodas pa vidu. Metamorfēšana izmanto iepriekš izrēķinātas atbilstības kartes un tādēļ ir ļoti efektīva. (8)

Metode ir balstīta uz to, ka attēlu kopa , no blakus skata punktiem ir ļoti koherenta(saistīta). Vairākums bilžu attēlo tos pašus objektus no nedaudz atšķirīgiem skatu punktiem. Šī metode izmanto kameras pozīciju un orientāciju un attēlu diapazona datus, lai automātiski atrast atbilstības starp attēliem. Atbilstības starp diviem blakus attēliem var būt iepriekš izrēķinātas un saglabātas ka metamorfēšanas karšu pāris. Izmantojot šīs kartes atbilstoši pikseļi tiek interaktīvi interpolēti, lai veidotu starp-attēlus. (8)

Šī metode strādā labi tad, kad ieejas skati atrodas tuvu viens otram, tādā veidā redzamības neskaidrības nerada īpašu problēmu. Citos gadījumos plūsmu laukus jāierobežo, lai nerastos pārklāšanas. Pie tam, kad divi skati atrodas tālu viens no otra attēlu daļas, kas pārklājas var kļūt pārāk mazas. Čena un Viljamsa metode strādā īpaši labi, kad visiem ieejas attēliem ir kopīgs skata virziens un izejas attēliem ir ierobežots skata leņķis, kas ir mazāks par 90 grādiem. (1)

Att. 1.2.1 Skatu interpolācijas rezultāti (8)

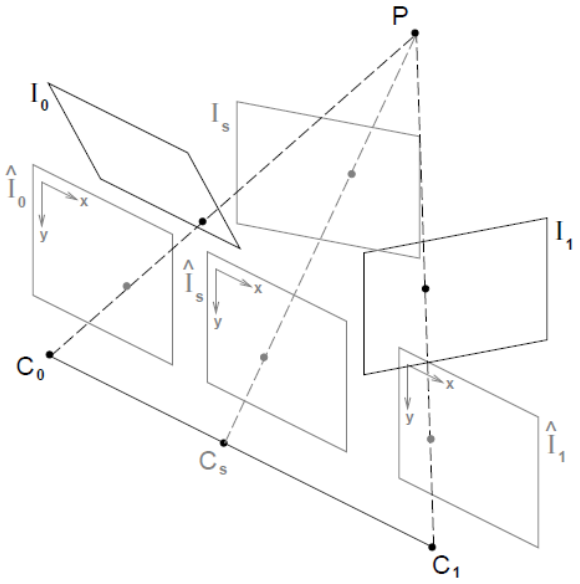
Izveidot plūsmas laukus skatu interpolācijai var būt grūti, it īpaši reālajiem attēliem. Jāizmanto datorredzes metodes, tādas kā iezīmju atbilstības vai stereo. Sintētiskiem attēliem plūsmu lauki var būt izrēķināti no dziļuma vērtībām. (1)

1.2.2 Skatu metamorfēšana

Skatu metamorfēšana no diviem attēliem rekonstruē skatu punktus, kas atrodas uz līnijas, kas savieno oriģinālu attēlu optiskus centrus.

Skatu metamorfēšana izmanto attēlu metamorfēšanu, rezultāta dabūjot formas, krāsas un pozas vienlaicīgo transformāciju, kas izskatās ļoti reālistiski. Skatu metamorfēšanas pirmais solis ir pirms-deformācija, pēc tam seko metamorfēšana, aiz kuras seko post-deformācija, kas dod beidzamo rezultātu.(7)

Ja netiek piesaistīta īpaša uzmanība, attēlu interpolācija izmantojot attēlu metamorfēšanu neveido nelokāmu 3D objektu transformācijas. Tāpēc ir nepieciešams veikt pirms-deformāciju un post-deformāciju.(7)



Att. 1.2.2 Skatu metamorfēšanas princips (7)

Starp-skati ir oriģinālu skatu lineāras kombinācijas tikai tad, ja kameras kustība asociēta ar starp-skatiem ir perpendikulāra kameras skata virzienam. Lai to saprastu, pieņemsim, ka projekciju matricas diviem paraugu skata punktiem ir P1 un P2. Nezaudējot vispārīgumu mēs varam pieņemt, ka P0 =M0(I|0) un P1 = M1(I|p), kur I ir 3x3 identitātes matrica un M0 un M1 ir matricas ar kameras iekšējiem parametriem(iekšējas kalibrēšanas matricas)

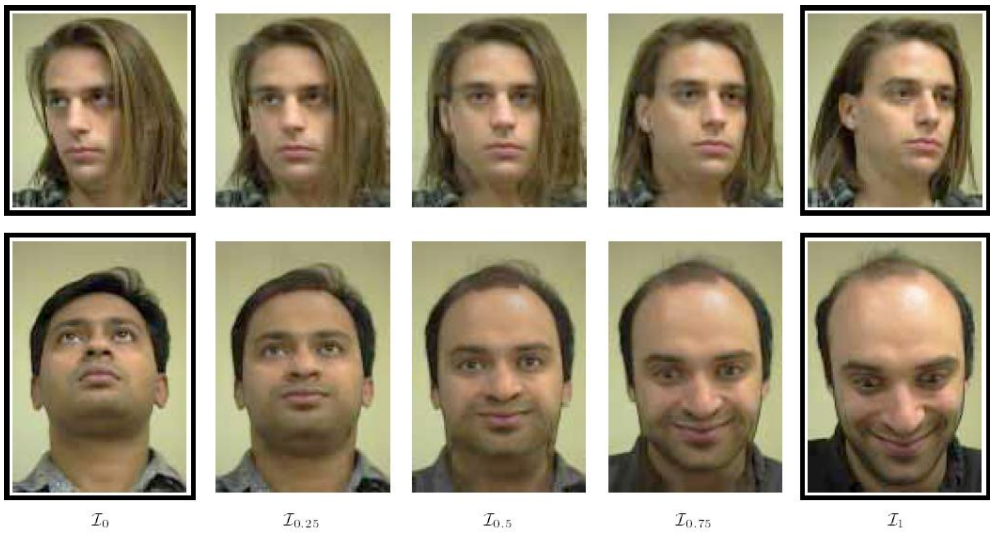
$$M_i = \begin{pmatrix} f_i & s_i & q_{xi} \\ 0 & a_i f_i & q_{yi} \\ 0 & 0 & 1 \end{pmatrix}$$

kur, f_i ir fokusa attālums, a_i - malu attiecības, s_i - slīpums un (q_{xi}, q_{yi}) galvenais punkts, p=$(p_x, p_y, 0)$ ir relatīva kameras kustība. Svarīgi ir tas, ka punkta p z komponente ir 0, tas ir atslēga attēlu linearitātei. (1)

Dots 3D punkts $x=(X, Y, Z, 1)^T$ tiek projicēts uz *u0 = 1/Z*P0*x* skatā 0 un *u1 = 1/Z*P1*x* skatā 1. Pieņemsim, ka mēs lineāri interpolējam 2D pozīciju virtuālajā skatā *It*, kā *ut = (1-t)u0 + tu1, 0<=t<=1*. Interesanti ir tas, ka *ut = 1/Z*Pt*x*, un *Pt = (1-t)P0 + tP1*, kas ir derīga(bet virtuāla) iekšējas kalibrēšanas starp-matrica. Ka rezultāts, paralēliem avota skatiem, ir fiziski korekti pielietot lineāru interpolāciju punktu pozīcijām, ja attiecības starp tiem ir pareizas. (1)

Ja divi avota attēli nav paralēli, tad var pielietot pirms-deformāciju, lai koriģētu tos tā, lai atbilstošas skenēšanas līnijas būtu paralēlas. Attiecīgi pēc-deformācija var tikt izmantota,

lai starp-skatu pataisītu par neparalēlu. Pie tam viss tas ir iespējams bez pilnīgi kalibrētas kameras. (1)



Att. 1.2.3 Skatu metamorfēšanas piemēri (7)

1.2.3 Pārsūtīšanas metodes

Pārsūtīšanas metodes raksturo tas, ka tās izmanto samēra nelielu attēlu kopu un pielieto tai ģeometriskus ierobežojumus, kas var būt iepriekš zināmi vai izrēķināti, lai projicētu attēla pikseļus konkrētajā skatu punkta. Ģeometriskie ierobežojumi var būt dažādā veida, gan zināmas dziļuma vērtības katrā pikselī, gan epipolārie ierobežojumi divu attēlu starpā, gan trifokalie/trilineārie tenzori, kas saista atbilstības starp attēlu tripletiem. Skatu interpolācija un skatu metamorfēšana būtībā arī pieder pārsūtīšanas metožu klasei. (1)

1.2.3.1 Fundamentāla/Pamata matrica

Laveau un Faugeras izmanto attēlu kolekciju, ko sauc par atsauces skatiem, un fundamentālas matricas principu, lai veidotu jaunus mākslīgus skatus. Fundamentāla matrica F ir 3x3 matrica ar rangu 2. Ja $x_1$un x_2 ir atbilstoši punkti skatos I_1 un I_2, tad $x_1^T F_{12} x_2 = 0$. x_2 atrodas uz epipolāras līnijas, kas var būt izrēķināta ka $F_{12}x_1$. Cits svarīgs termins ir epipols, visas epipolaras līnijas krustojas epipolā, un tas ir citas kameras projekcijas centra projekcija. Epipols e_{12} skatā divi ir "nulles telpa", kodols matricai F_{12}, t.i. $F_{12}e_{12} = 0$. (1)

Pieņemsim ka punktu atbilstības un fundamentāla matrica attēlu pārim tika iegūti. Virtuālās kameras skatu punkts (skats 3) tiek noteikts ar lietotāja palīdzību, lietotājs izvēlas divus punktus e_{13} (attēls 1) un e_{23} (attēls 2) tā, lai $e_{23}^T F_{12} e_{13} = 0$. Attēla plakne, kas ir saistīta ar skatu 3, tad tiek interaktīvi izvēlēta norādot 3 atbilstošu punktu pārus plus vienu punktu vienā no attēliem. Pēdējo punktu nav nepieciešams izvēlēties manuāli otrajā attēlā, jo tas var būt automātiski izrēķināts izmantojot kolinearitāti un epipolārus ierobežojumus. (1)

Lai izvairītos no caurumiem, jauns skats tiek aprēķināts izmantojot apgriezto kartēšanu vai staru trasēšanu. Katram pikselim m_3 jaunajā mērķa attēlā tiek veikta meklēšana, lai atrastu atbilstību pāri atsauces skatos. Katram i pikselim m_{1i} epipolāras līnijas garumā skatā 1 mēs pārbaudām vai tam atbilstošais punkts m_{2i} skatā 2 apmierina epipolāru ierobežojumu $m_{2i}^T F_{32} m_3 = 0$. Citiem vārdiem sakot, mēs meklējam pa $F_{31}m_3$, kamēr līkne cor($F_{31}m_3$) un $F_{32}m_3$ nekrustojas. Pikselis tiek pārsūtīts, ja tāds krustošanas punkts ir atrasts. (1)

Ir svarīgi piezīmēt, ka ja kamera ir vāji kalibrēta, tad atgūtam skata punktam būs projektīva struktūra. Tas notiek tāpēc, ka eksistē 3D projekciju un struktūru klase, kas radīs vienādus sākuma attēlus. Dēļ tā, ka leņķi un laukumi netiek saglabātas, rezultāta skata punkts var izskatīties deformēts. Zinot kameras parametrus var izvairīties no šīm problēmām. (1)

1.2.3.2 Trifokālais tenzors

Fundamentāla matrica izveido attiecības starp diviem taisniem skatu punktiem bez jebkādas informācijas par scēnas struktūru, savukārt trifokālais(trilineārais) tenzors izveido projektīvas attiecības starp trim skatiem. Trifokālais tenzors ir 3x3x3 matrica kas satur informāciju par punktu un līniju attiecībām starp trim skatiem un fundamentālas un projekciju matriču ieguvi. (1)

Ja attēlu tripletam ir zināms trifokālais tenzors, tad, ja ir dota divu punktu atbilstība divos attēlos no tā tripleta, trešais atbilstošais punkts trešajā attēlā var būt tieši izrēķināts bez 3D[projekciju] rēķināšanas. Divu attēlu epipolārā meklēšanas metode, kas tika iepriekš apskatīta nestrādā pareizi, kad divas epipolāras līnijas virtuālajā attēlā sakrīt. Trifokālais tenzors izvairās no šī gadījuma, dēļ elastīgas dabas attiecībām starp punktiem un līnijām trijos skatos. (2) Piemēram, pieņemsim, ka ir nepieciešams izrēķināt punktu m3 trešajā skatā, ir doti punkti m1 un m2 pirmajā un otrajā skatā un trifokālais tenzors $\mathcal{T}$, kur elements (i, j, k) tiek pierakstīts $\mathcal{T}_j^{ik}$. Mēs varam atrast līniju l_2, kas ir perpendikulāra epipolārai līnijai, ko dod $F_{21}m_2$, pēc tam m_3 var tikt atrasts izmantojot attiecību $(m_3)^k = (m_1)^i (l_2)_j \mathcal{T}_j^{ik}$. (1)

Trifokāla tenzora punktu pārsūtīšanas īpašība tika izmantota jaunu skatu ģenerēšanai no diviem vai trijiem ieejas attēliem. Jaunu skatu ģenerēšanas ideja ir diezgan vienkārša. Vispirms no punktu atbilstībām starp ieejas attēliem tiek izrēķināts trifokālais tenzors. Divu attēlu gadījumā viens attēls tiek dublēts un uzskatīts par trešo. Tiek pieņemts ka kameras iekšēji parametri ir zināmi, kas atvieglo jauna skata atrašanu. Trifokālais tenzors, kas ir piesaistīts trešajam skatam var būt izrēķināts no zināmam pozas izmaiņām(izmaņas rotācijā vai translācijā) attiecībā pret trešās kameras pozīciju. Kad tirfokālais tenzors un atbilstības starp diviem ieejas attēliem ir zināms, punkti var būt pasūtīti izmantojot tiešo kartēšanu(pikseļu nodošana no ieejas attēliem uz virtuāliem skatiem). Nav skaidrs ka tiek apstrādāta redzamība, bet, tam var būt izmantots modificēt "mākslinieka"(prioritārās aizpildīšanas) algoritms. Papildus var tikt izmantota splatēšana, tur, kur ieejas attēla pikselis tiek kartēts vairākos pikseļos, lai novērstu caurumu parādīšanos jaunos skatos. (1)

1.2.4 Secinājumi

Metodes, kas izmanto neizteiktu skatuves ģeometriju, izmanto mazāku avota attēlu skaitu un tādējādi prasa mazāk no fiziskās atmiņas resursiem. Turklāt viņi izmanto ģeometriskās metodes, kas ļauj, balstoties uz atbilstībām starp attēliem, iegūtu vajadzīgo transformāciju, karu vēlāk var pielietot sākotnējam attēlam, lai radītu jaunus skatus. Galvenais metožu pielietošanas veids - jaunu skatu vai starp-skatu iegūšana.

1.3 Renderēšana ar izteiktu ģeometriju

Metodes, kas nebalstās uz ģeometriju, parasti pieprasa daudz attēlu paraugu renderēšanai un metodes kas izmanto neizteiktu ģeometriju pieprasa precīzu attēlu reģistrāciju augstas kvalitātes skatu sintēzei. Šinī sadaļā tiks aprakstītas metodes, kas izmanto izteiktu ģeometriju. Šīm metodēm ir pieejama tieša 3D informācija par scēnu, 3D koordinašu vai dziļuma zināmas skata līnijas garumā formā. Tradicionālais 3D modelis ar vienu faktūrkarti var tikt uzskatīts, ka šīs klases īpašais gadījums. (1)

Metodes, kas izmanto izteiktu ģeometriju ietver sevī stendus, gariņus, reljefa faktūras, slāņotus dziļuma attēlus un no skata atkarīgas faktūru kartes. Gariņi var būt plakani vai tiem var būt patvaļīga dziļumu sadale, jauni skati tiek veidoti izmantojot 3D deformācijas. LDI paplašina metodi ka pie kārto dziļumu katram pikselim, tādēļ ka LDI var iekodēt vairākus dziļumus katram konkrētajam staram. No skata atkarīgas faktūrkartēšanā ir vairāku faktūru kartēšana uz vienas un tas pašas 3D virsmas, kur krāsām tiek atrasts vidējais izmantojot svarus, kas ir balstīti uz virtuāla skata punkta tuvumu ieejas skata punktam. (1)

1.3.1 3D deformācijas

Gadījumā, kad katram punktam vienā vai vairākos attēlos ir pieejama dziļuma informācija, tuvu skatu punktu renderēšanai var tikt pielietotas 3D deformācijas tehnikas. (9)

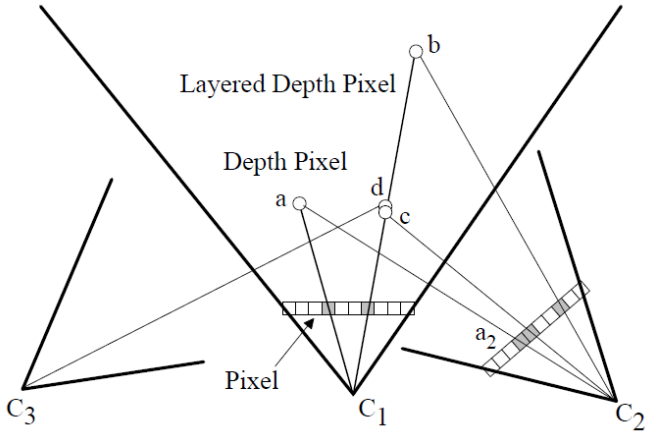
Attēls var tikt norenderēts no jebkura tuva skatu punkta, projicējot oriģināla attēla pikseļus to 3D koordinātēs un uzprojicējot tos jaunajā attēlā. Vislielākā problēma 3D deformāciju metodē ir caurumi deformētajā attēla. Caurumi rodas dēļ ieejas un izejas attēlu paraugu ņemšanas izšķirtspējas starpības un dēļ disoklūziju, kur scēnas daļa ir redzama izejas attēla, bet nav redzama ieejas attēlos. Caurumu aizpildīšanai izmanto splatēšanu/traipināšanu. (1)

Lai uzlabotu 3D deformāciju renderēšanas ātrumu, deformēšanas process var tikt sadalīts divos samērā vienkāršos soļos, pirms-deformēšanā un tradicionālā faktūrkartēšanā. Faktūrkartēšanu var veikt standartā grafikas aparatūra.
3D deformāciju metode var tikt pielietota ne tikai tradicionālas perspektīvas attēliem, bet arī multi-perspektīviem attēliem. (1)

1.3.2 Slāņoti dziļuma attēli

Lai risinātu problēmas, kas 3D deformāciju metodē ir saistītas ar disoklūziju, Šeids (10) piedāvāja slāņotus dziļuma attēlus(LDI), tie glabā ne tikai redzamo ieejas attēlā, bet arī to, kas atrodas aiz redzamas virsmas. Viņa rakstu darbā, LDI tiek konstruēti vai nu no

stereo attēlu kopas, ar zināmu kameras kustību vai tieši no mākslīgām vidēm ar zināmu ģeometriju. LDI katrs ieejas attēla pikselis satur sevī sarakstu ar dziļuma un krāsas vērtībām, kas atbilst tiem punktiem, kas veidojas, kad stars no pikseļa šķērso apkārtējo vidi. (1)



Att. 1.3.1 Slāņots dziļuma attēls (10)

Kaut gan LDI pietiekoši vienkārši ļauj deformēt vienu attēlu, tas nepievērš uzmanību paraugu blīvuma jautājumam. Čengs piedāvāja LDI kokus, lai paraugu ņemšanas norma ieejas attēliem tiktu saglabāta, katram pikselim adaptīvi izvēloties LDI LDI kokā. Renderējot LDI koku tikai koka līmenim, kas ir salīdzināms ar paraugu ņemšanas normu izejas attēla, jābūt izietam cauri. (1)

1.3.3 No skata atkarīgas faktūru kartes

Faktūru kartēs tiek plaši pielietotas datorgrafikā fotoreālistisko vižu ģenerēšanai. Mākslīgai videi faktūrkartēti modeļi var tikt ģenerēti izmantojot CAD modelēšanas rīku. Reālajai videi šie modeļi var tikt ģenerēti izmantojot 3D skeneri vai pielietojot datorredzes metodes saņemtiem attēliem. Diemžēl datorredzes metodes nav pietiekoši spēcīgas, lai atgūtu precīzus 3D modeļus. Pie tam, izmantojot vienu faktūrkartētu modeli, ir grūti dabūt vizuālus efektus tādus, ka izgaismotas vietas, atstarojumus un caurspīdīgumu. (1)

Lai dabūt šos vizuālus efektus no rekonstruētas arhitektūras vides, Debevecs izmanto no skatu atkarīgu faktūrkartēšanu jaunu skatu renderēšanai, deformējot un sastādot kopā vairākus vides ieejas attēlus. Tas atbilst parastai faktūrkartēšanai, atšķiras tas, ka vairākas faktūras no dažādiem paraugu skatu punktiem tiek deformētas uz vienu un to pašu virsmu un tiek atrasts to vidējais, kur svari tiek izrēķināti balstoties uz tekoša skatu punkta tuvumu piemēru skatu punktiem. Vēlāk Debevecs piedāvāja trīs-etapu no skatu punktu atkarīgu faktūrkartēšanas metodi, lai samazinātu skaitļošanas cenu un dabūt gludākas pārejas. Šī metode izmanto redzamības pirmapstrādi, poligonu skatu kartes un projektīvo faktūrkartēšanu. (1)

1.3.4 Secinājumi

Metodēm, kas izmanto izteiktu skatuves ģeometriju ir acīmredzamas priekšrocības salīdzinājumā ar citām, tādā nozīmē, ka papildus scēnas attēliem, viņiem ir piekļuve precīzai ģeometriskai informācijai. Pateicoties tam, avota attēlu skaitu var dažos gadījumos samazināt līdz vienam. Pie tam, virtuālo attēlu kvalitāti, tas īpaši neietekmē. Trūkums ir tas, ka iegūt precīzu ģeometrisko informāciju par scēnu ne vienmēr ir iespējams un tas var būt saistīts ar daudziem sarežģījumiem. Pie tam, attēli, kas tiek iegūti izmantojot šo metodi, ne vienmēr it tik pat fotoreālistiski, kā attēli, kas tiek iegūti ar metodēm, kas nelieto scēnas ģeometriju.

Viens metožu pielietošanas veids - starp-skatu veidošana / renderēšana, grafisko sistēmu darbības paātrināšanai. Pēc manām domām metodes, kas pieder pie šīs klases, ir vairāk piemērotas gadījumos, kad sākotnējie attēli tiek radīti mākslīgi.

2 Praktiskā daļa

2.1 Problēmas apraksts

Ir doti divi attēli, kas tika nofilmēti dažādos laika momentos no viena un tā paša skatu punkta, uzdevums ir rekonstruēt šo divu attēlu starpattēlu, t.i. bildi, kas attēlotu apkārtējo vidi laika momentā starp laika momentiem, kuros tika veidoti ieejas attēli. Attēli, tika uzņemti, izmantojot vienu un to pašu kameru.

2.2 Risinājuma detalizētais teorētiskais apraksts

Ir pieejami divi plenoptiskas funkcijas paraugi dažādos laika momentos, paraugi apraksta tikai daļu no plenoptiskās funkcijas, līdz ar to funkcija ir nepilnīga. Attiecībā uz kameru, tā paliek nekustīga abos laika momentos. No tā izriet, ka mums ir nepilnīga plenoptiska funkcija ar argumentiem: koordinātes, krāsa un laiks. Šis gadījums atšķiras no teorētiskajā daļa aprakstītiem ar to, ka šinī gadījuma netiek izmantoti vairāki skatu punkti, bet attēli no viena skatu punkta vairākos laika momentos. Tas nozīme, ka mūsu vide nav statiska. Tāpēc plenoptiskā funkcija šīm gadījumam izskatīsies sekojoši:

$$\mathrm{P} = \mathrm{P}(\theta, \phi, \lambda, t)$$

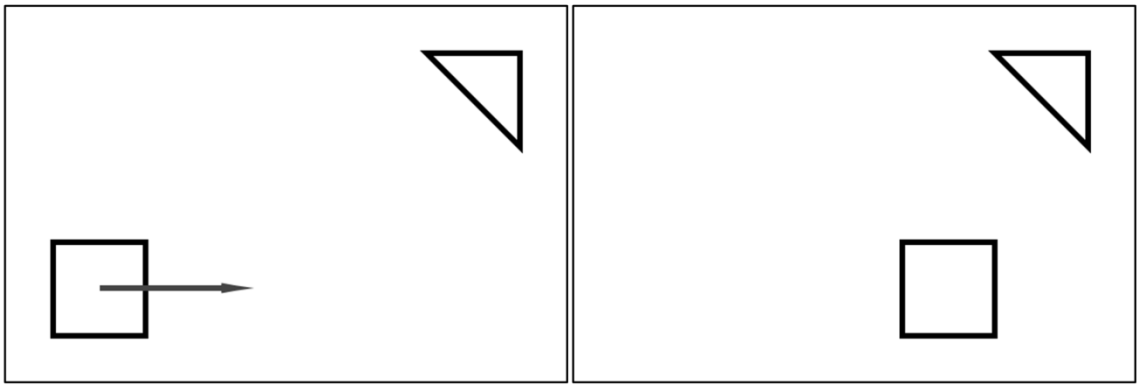
Melnbaltajam attēlam:

$$\mathrm{P} = \mathrm{P}(\theta, \phi, t)$$

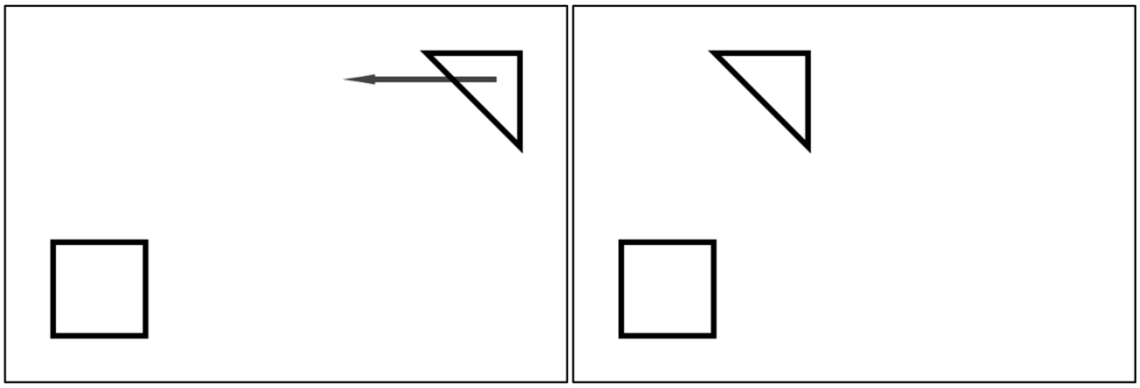
Šī funkcija satur pietiekami daudz informācijas, lai izveidotu starp-attēlu, bet tā ir neērtā mūsu mērķiem veidā, jo neviena no iepriekš aprakstītām metodēm, neizmanto, kā plenoptiskas funkcijas paraugus, paraugus ņemtus dažādos laika momentos. Visas iepriekš izskatītas metodes izmanto attēlu kopas, kas tika uzņemti vienā laika momentā, no dažādiem skatu punktiem. Tāpēc, ja mēs nepārveidosim esošo funkciju, mēs nevarēsim izmantot nevienu no iepriekš aprakstītām metodēm.

Apskatīsim izmaiņas, kas rodas laika intervālā Δt starp attēlu uzņemšanām. Gadījumā, ja novērošanas objekts ir ceļš, laika intervālā Δt var mainīties to objektu koordinātes, kas pārvietojas pa ceļu. Ja mēs pieņemam, ka plakne, ko veido koordinātu taisnes X un Z ir paralēla zemei, un Y koordinātu taisne ir perpendikulāra tai, tad kustīgiem objektiem var mainīties koordinātes X un Z. Izmaiņas, Y koordināti var neņemt vērā, jo pacelties gaisā automašīnas nevar. Teorētiski, pārvietojot objektu attiecībā pret kameru par noteiktu vektoru ir tas pats, kas pārvietot kameru attiecībā pret objektu par to pašu vektoru, kas ir vērsts pretējā virzienā. Objekta projekcija uz kameras plakni būs identiska abos gadījumos (Att. 2.2.1, Att. 2.2.2). Tas dod mums iespēju pieņemt, ka mums ir nevis divi attēli dažādos laika momentos, bet divi attēli vienā laika momentā no dažādiem skata punktiem. Faktiski, tā tas būtu, ja visi, uz scēnas esošie, objekti pārvietotos par vienu un to pašu vektoru, bet reālajā pasaulē visi objekti, kas atrodas uz scēnas pārvietojas ar savu ātrumu un laika intervālā Δt pārvietojas atšķirīgi viens no otrā, tāpēc katram objektam būs savs kustības vektors. Šo problēmu var atrisināt, sadalot objektus uz scēnas grupās,

pēc kustības vektora, un katrai grupai piešķirot savu kameru. Tādējādi otro attēlu var uzskatīt par dažādu objektu projekciju kombināciju uz atbilstošam kamerām.

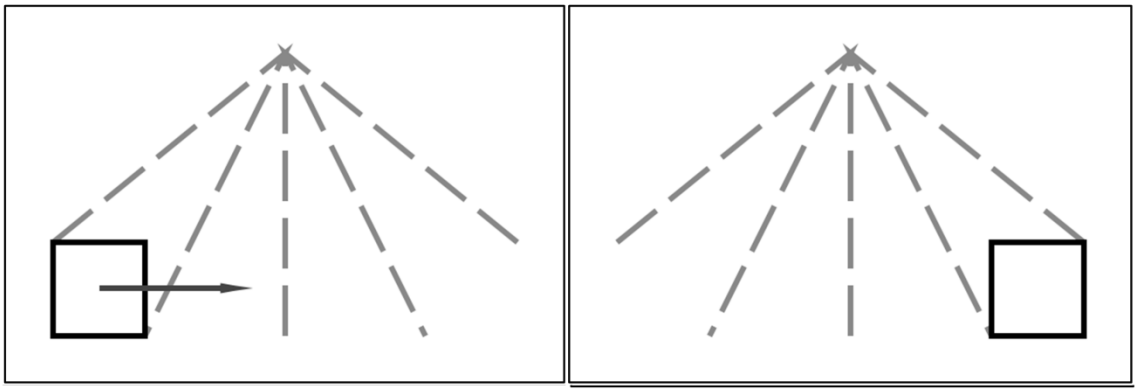


Att. 2.2.1Kustās objekts

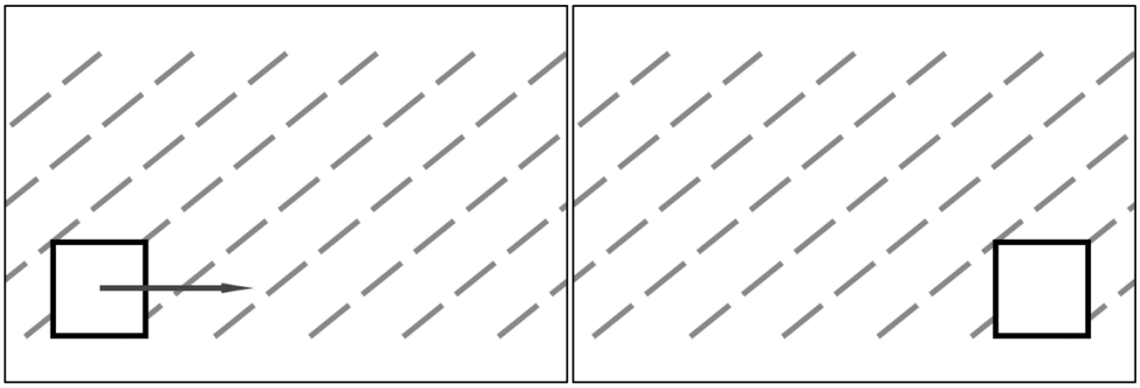


Att. 2.2.2 Kustās kamera

Līdz šim mēs apskatījām izmaiņas projekcijās, bet reālajā pasaulē ir vēl viens šķērslis, kas neļauj veikt pāreju no vienas parametrizacijas uz citu. Lai mēs bez sekām varētu aizvietot objektu pārvietošanu uz kameras pārvietošanu, ir nepieciešams, lai visi pārējie objekti uz scēnas pārvietotos kopā ar kameru, ieskaitot gaismas avotus. Ja kameras kustības laikā, gaisma paliek uz vietas, tad rezultātā attēls var stipri atšķirties no tā, kas būtu gadījumā, kad pārvietojas objekts (Att. 2.2.3). Ja objekta apgaismojums abos stāvokļos ir līdzīgs, piemēram, ja gaismas avots atrodas lielā attālumā, kas ir daudz lielāks nekā attālums, par kuru objekts ir pārvietojies, tad mēs varam pieņemt, ka pārvietojas nevis objekts, bet kamera (Att. 2.2.4). To var izdarīt, jo, ja gaismas avots ir ļoti lielā attālumā, starp stariem, kas sasniedz objektu, būs ļoti mazs leņķis. Tā kā ceļi parasti ir atklātās teritorijās, un galvenais gaismas avots ir saule, vairumā gadījumu, gaismas avotiem nevajadzētu radīt problēmas.



Att. 2.2.3 Tuva gaismas avota stari



Att. 2.2.4 Tāla gaismas avota stari

No iepriekšējiem pārdomām seko, ka mēs varam pārveidot doto funkciju funkcijā, kurai būtu mums ērtāks izskats, kuras argumenti ir punkta koordinātes, krāsa un kameras koordinātes. Tas dod mums iespēju izmantot kādu no iepriekš aprakstītām metodēm, lai atrastu starpposma attēlu.

Tagad ir nepieciešams izvēlēties kādu metodi, kas ir vislabāk piemērota, problēmas atrisināšanai. Mums ir tikai divi attēli, tāpēc pirmās grupas metožu realizācijai mums ir nepietiekami daudz scēnas paraugu. Un trešās grupas metožu realizācijai mums nav pietiekami daudz informācijas par scēnas ģeometriju. No tā izriet, ka no iepriekš aprakstītām metodēm, vislabāk mums der metodes, kas pieder otrai grupai, jo tās ļauj mums iegūt starpposma rezultātus, balstoties uz nelielu attēlu skaitu, un neprasa ģeometriskās informācijas par scēnu. Tagad ir jāizvēlas viena no otrās grupas metodēm. Principā visas otrās grupas metodēm izmanto vai nu fundamentālo matricu (bifokālais tenzors), vai trifokālo tenzoru. Trifokālo tenzoru var iegūt no diviem attēliem, izmantojot kādu no ieejas attēliem trešā attēla vietā, bet, protams, rezultāts būs labāks, ja izmanto trīs dažādus attēlus. Tāpēc es domāju, ka diviem attēliem ir labāk izmantot metodi, kas balstās uz bifokāla tenzora. No palikušajam metodēm problēmas risināšanai vislabāk piemērota ir skatu metamorfēšana. Skatu metamorfēšana nenodarbojas ar patvaļīgu ainas skatu atrašanu, atšķirībā no daudzām citām metodēm, šī metode ar to, ka ģenerē pārejas starp diviem attēliem, tā ir paredzēta, lai ģenerētu skatus, kas atrodas uz līnijas, kas savieno attēlu optiskus centrus. Tas ir tieši tas kas ir nepieciešams, lai atrisinātu doto problēmu.

Skatu metamorfēšanai nepieciešami divi attēli, kas parāda vienu un to pašu ainu, projekcijas matricas, kas atbilst šiem attēliem un atbilstības starp pikseļiem abos attēlos. Attēlu projekcijas matricas var atrast, izmantojot fundamentālo matricu, tāpēc mums ir nepieciešamas atbilstības starp diviem attēliem. Kā ir iespējams iegūt projekciju matricas no fundamentālas matricas, ir aprakstīts publikācija (7).

Lai paveiktu metamorfēšanu no diviem attēliem mums ir nepieciešams vispirms veikt pirmsdeformāciju abiem attēliem, lai abi scēnas skati būtu paralēli. Tas ir svarīgi, jo divu paralēlu skatu lineāra interpolācija neizjauc formu, tas tika pierādīts publikācijā (7). Ja skati nav paralēli, tad objektu forma pēc interpolācijas, varētu tikt izjaukta.

Nākamais posms - ir metamorfēšana. Metamorfēšanu var izdarīt, izmantojot vienu no attēlu metamorfēšanas metodēm, vai nu lineāri interpolējot pikseļu koordinātes un krāsas. Vienādojums parāda, kā noteikt starpvērtību rēķināšana funkcijai ar vienu argumentu ($0 \leq s \leq 1$):

$$F(x'') = (1 - s) * F(x) + s * F(x')$$

Pēdējais etaps - pēcdeformācija. Pēcdeformācija pārveido starpskatu no paralēla uz neparalēlo, pielietojot attēlam projekcijas transformāciju, ko iegūst, interpolējot divas avota attēlu projekciju matricas.

Es pamanīju, ka objekti mūsu gadījumā pārsvara kustās viena virzienā un es domāju, ka šinī gadījumā mēs varam izlaist pirmo un pēdējo skatu metamorfēšanas soli, jo projekcijas uz kameru šinī gadījumā būs paralēlas. Tas seko no tā, ka kameras kustība ir atkarīga no objekta kustības. Šinī gadījumā mēs varam metamorfēt divus ieejas attēlus pa tiešo.

Gribu piebilst, ka izmantojot interpolācijas metodi ir iespējams dabūt ne tikai attēlu, kas ir tieši pa vidu, bet jebkuru attēlu no attēlu sērijas, kas savieno ieejas attēlus.

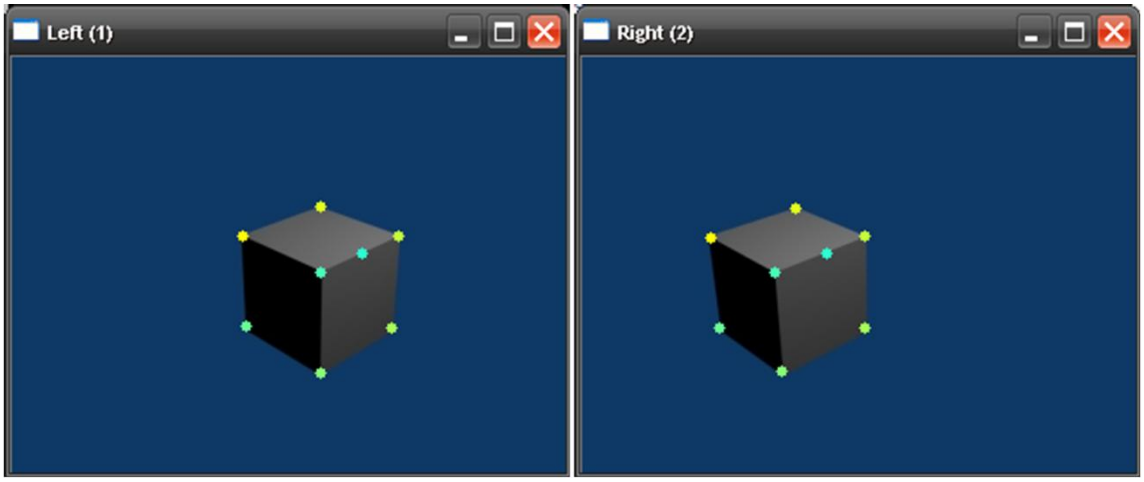
2.3 Realizācijas apraksts

Lai īstenotu risinājumu, es izmantoju valodu C + + ar bibliotēku OpenCV. Bibliotēkā, jau bija man nepieciešami līdzekļi, kas ļāva man nerakstīt visu kodu no nulles, un būtiski paātrināja risinājuma izstrādi, kaut gan tie līdzekļi skaitījās, ka eksperimentāli. Tas ļāva man koncentrēties uz risinājuma loģikas.

Pirmais solis ir ielādēt abus attēlu programmā. Attēli var būt krāsaini, vai melnbalti. No attēla izmēriem ir atkarīgs algoritma izpildes laiks. Attēlu formāts var būt *. bmp, *. jpg, *. png.

Nākamais solis ir atbilstošu punktu atzīmēšana abos attēlos. Šī daļa tiek pildīta manuāli. Es eksperimentēju ar iespēju automātiski atrast attiecīgus punktus abos attēlos, bet rezultāti nebija apmierinoši, tāpēc es nolēmu, ka var ierobežoties ar manuālu punktu izvēli. Manā programmā var izvēlēties galvenos punktus divos un vienā attēlā. Ja punkti ir izvēlēti tikai vienā attēlā, atbilstoši punkti otrajā attēlā tiek atrasti, izmantojot Lukasa-

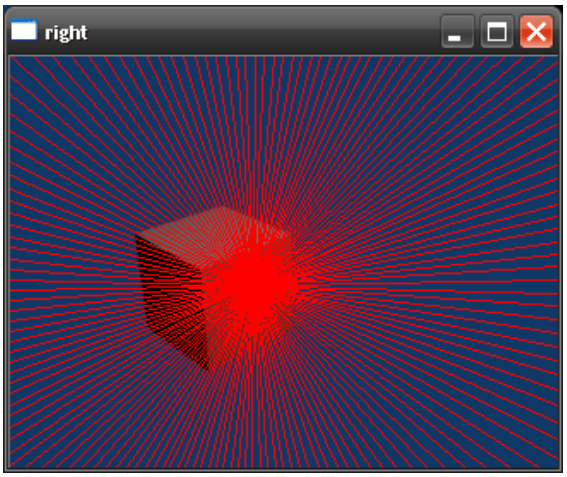
Kanades piramīdu algoritmu, kas netiks apskatīts, jo attiecas uz datorredzēs jomu un iziet ārpus šī darba robežām. Ja atbilstoši punkti ir zināmi iepriekš, tad tos nav jāievada manuāli katru reizi, tie var būt ielādēti programmā automātiski.



Att. 2.3.1 Atbilstoši punkti abos attēlos

Ja attēli un punkti ir ielādēti, mums ir viss nepieciešamais, lai renderētu starpposma skatus. Tagad, pamatojoties uz pieejamo informāciju, ir nepieciešams atrast fundamentālo matricu, tā kalpos ka pamats visiem turpmākajiem aprēķiniem. Kā jau tika aprakstīts iepriekš, fundamentāla matrica savieno divus punktus tā, lai $x_1^T F_{12} x_2 = 0$. No tā izriet, ka, balstoties uz atbilstībām starp punktiem, mēs varam atrast fundamentālu matricu, atrisinot vienādojumu sistēmu.

Kad fundamentāla matrica ir aprēķināta, var aprēķināt visas epipolāras līnijās, kuras pieder abiem attēliem. Katram attēlam, tiek aprēķināta epipolāru līniju grupa tā, lai līnijas pilnībā pārklātu attēlu.



Att. 2.3.2 Epipolāras līnijas

Vēlāk interpolējot šīs līnijas, mēs varēsim iegūt jaunu attēlu. Kad mums ir zināmas epipolāras līnijas, mums ir nepieciešams, veikt pirmsdeformāciju abiem attēliem.

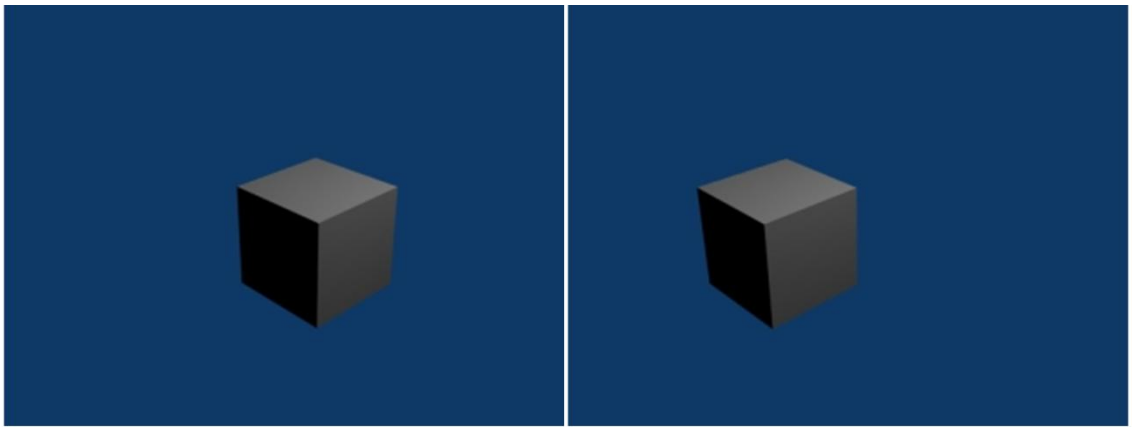
Pēc tam ir nepieciešams uzzināt, kuras rindās atbilstoši kuram. Lai to izdarītu, katrai līnijai tiek piešķirta pikseļu grupa, kas atrodas uz šīs līnijas. Pēc tam, starp pikseļu grupām tiek meklētas atbilstības, kas būs nepieciešamas, lai veiktu metamorfēšanu.

Tad epipolāras līnijas tiek interpolētas, lai iegūtu epipolāras līnijas starpposma skatam. Pēc tam attēli tiek metamorfēti un rezultāta attēls tiek pēcdeformēts. Pēc tām rezultāta attēls tiek rādīts atsevišķā logā.

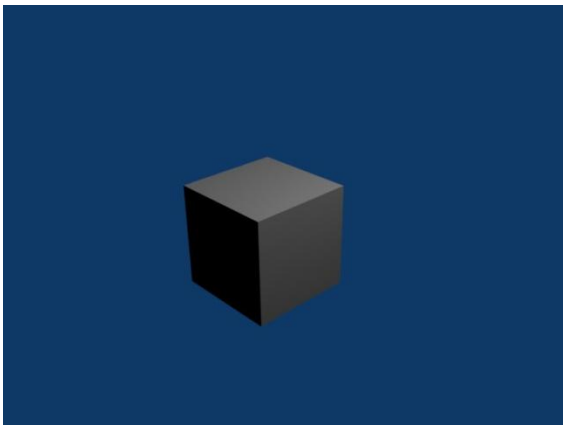
Līdz tam es savā darbā secināju, ka tad, kad objekti nemaina kustības virzienu, varat izlaist pirmsdeformāciju un pēcdeformāciju un tieši metamorfēt attēlus. Lai to pārbaudītu, es atsevišķi no iepriekšējā procesā interpolēju programmai padotus punktus, lai atrastu starpposma punktu komplektu. Tad es meklēju transformācijas matricas, kas var pārveidot sākuma punktus, punktos kas tika atrasti ar interpolācijas palīdzību un pielietoju šīs matricas sākotnējiem attēliem. Tad es samazinu alfa kanālu abiem attēliem un izlieku vienu otram virsū. Būtībā šis process ir līdzīgs metamorfēšanai, ja izpildās attiecīgi nosacījumi.

2.4 Rezultāti

Rezultātā, man sanāca izveidot programmu, kas atrod starpposma skatu, bet, diemžēl, attēla kvalitāte neatbilst avota attēlu kvalitātei. Uz bildēm (Att. 2.4.1, Att. 2.4.2) ir attēlota aina, kas tika radīta 3D modelēšanas programmā Blender, pirmie divi attēli tiek padoti programmai, kā ieejas dati, un trešais ir attēls, ko programma cenšas atrast.

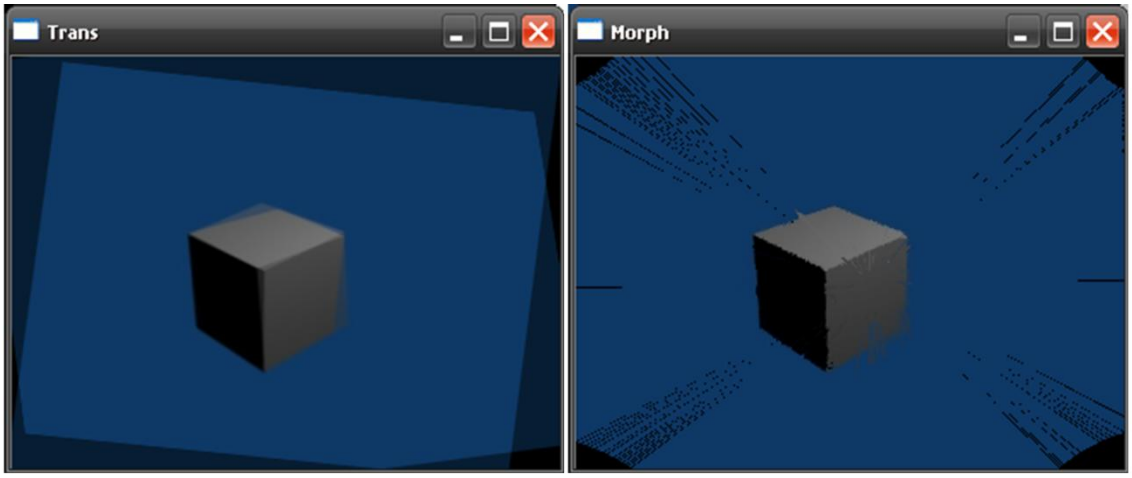


Att. 2.4.1 Ieejas attēli

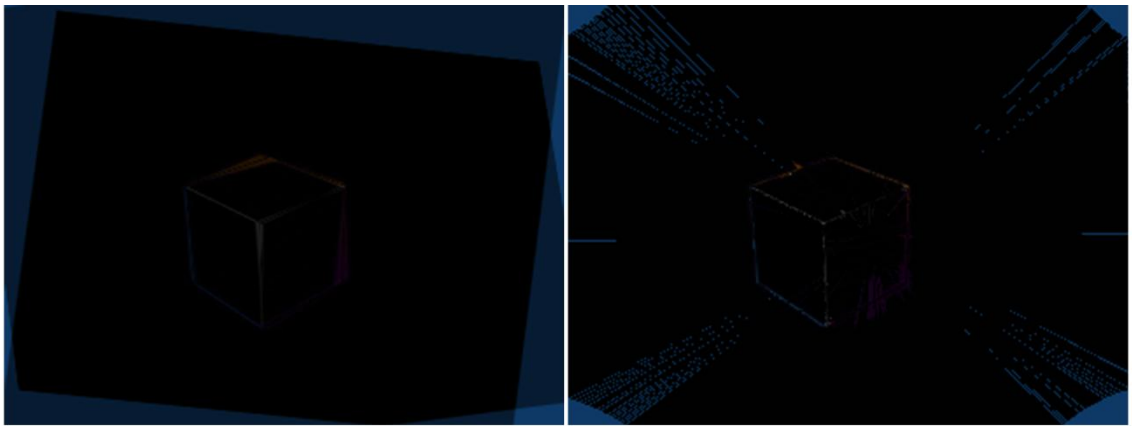


Att. 2.4.2 Etalons

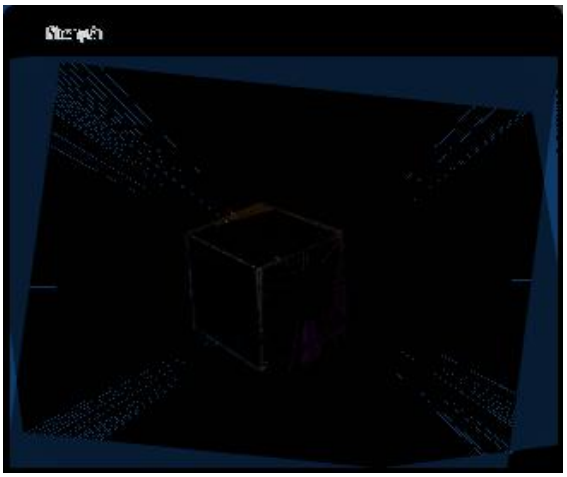
Bildē (Att. 2.4.3) parādīti manas programmas darbības rezultāti, kreisajā pusē attēls iegūts, izmantojot pārejas matricas, kas tika pielietotas sākotnējiem attēliem, attēls pa labi tika iegūts ar skatu metamorfēšanas metodi. Bildes (Att. 2.4.4, Att. 2.4.5) ir starpības starp attēliem, kas ir veidotas, lai labāk uzsvērtu atšķirības starp tiem. Pirmie divi parāda atšķirības starp rezultātiem un etalonu, un trešā parāda atšķirības starp rezultātiem. Kā varat redzēt, abi rezultāti ir diezgan tuvu etalonam, bet ir dažas atšķirības. Rezultāti arī savstarpēji ir diezgan līdzīgi.



Att. 2.4.3 Rezultāti virtuālai videi



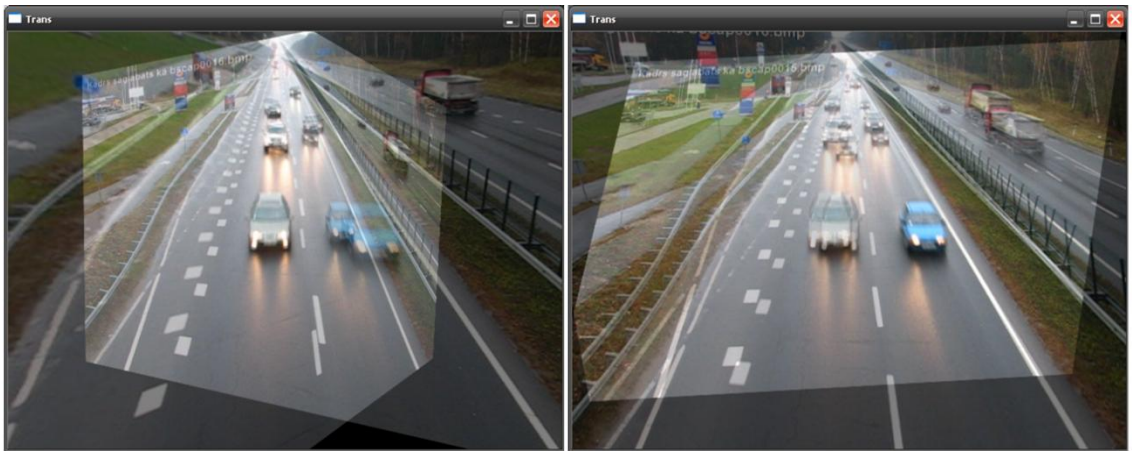
Att. 2.4.4 Rezultātu starpības ar etalonu



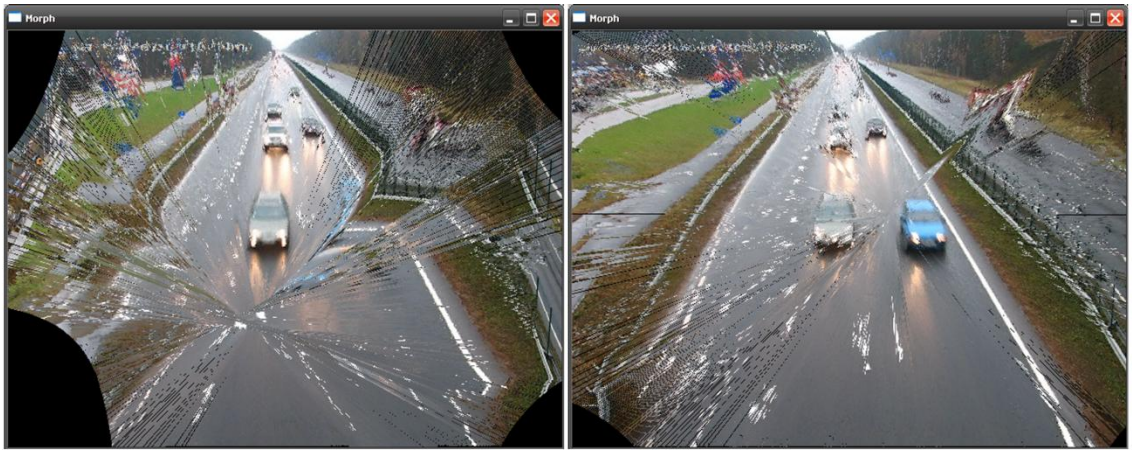
Att. 2.4.5 Rezultātu savstarpēja starpība

Diemžēl reāliem attēliem man nav etalona, ar kuru būtu iespējams salīdzināt rezultātus, tāpēc es salīdzināšu divus attēlus, kas tiks dabūti ar metamorfēšanas metodi un ar savu transformāciju metodi. Bildē (Att. 2.4.6) redzami sākotnēji attēli. Attēli, kas ir redzami bildē (Att. 2.4.7), tika iegūti izmantojot projektīvo transformāciju. Kreisais attēls atbilst kreisai automašīnai, labais labai.. Bildē (Att. 2.4.8) ir redzami attēli, kas tika iegūti, skatu metamorfēšanu, tām pašām automašīnām.

Att. 2.4.6 Ieejas attēli (reāla vide)

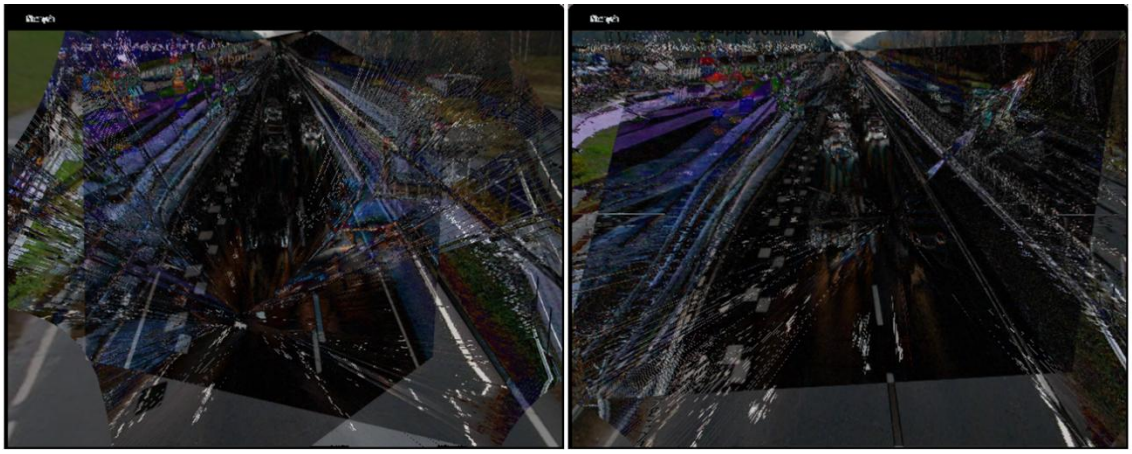


Att. 2.4.7 Projektīvo transformāciju rezultāti



Att. 2.4.8 Skatu metamorfēšanas rezultāti

Bildē (Att. 2.4.9) var redzēt attēlu, kas iegūti ar dažādām metodēm, atšķirības. Pirmais attēls attiecas uz auto pa kreisi, otrais uz auto pa labi. Kā redzams no attēliem, vieta, kur atrodas automašīna, ir gandrīz pilnīgi melna, tas norāda, ka starp attēliem atšķirības ir minimālas.



Att. 2.4.9 Rezultātu savstarpējas starpības

2.5 Secinājumi

Es realizēju skatu metamorfēšanas algoritmu, no, kura sagaidīju diezgan augstas kvalitātes rezultātus, diemžēl, daudzos gadījumos algoritma ražoto attēlu kvalitāte ir zemāka nekā oriģināliem attēliem. Es uzskatu, ka vainīgs ir tas algoritms, kas ir atbildīgs par attēlu metamorfēšanu. Man ir aizdomas, ka aizstājot to ar citu, varētu sasniegt daudz labākus rezultātus. Pārsteidzoši, ka rezultāts, kas iegūts no reāliem attēliem, bija augstākas kvalitātes, gan skatu metamorfēšanas algoritmam, gan manai eksperimentālai metodei. Bet kopumā es domāju, ka rezultāti ir diezgan labi, neskatoties uz kropļojumiem un neprecizitātēm. Es uzskatu, ka man izdevās pierādīt, ka izmantojot skatu metamorfēšanas metodi var iegūt starpposma attēlu diviem ceļa attēliem dažādos laika momentos.

3 Rezultāti

Padarīta darba rezultātā tika atrisināta problēma, kas tika nodefinēta darba sākumā. Problēma tika atrisināta, izmantojot vienu no metodēm, kas tika apspriestas teorētiskajā daļā.

Šajā darbā tika apspriestas dažādas metodes, kas tiek izmantotas uz attēliem balstītajā renderēšanā. Metodes, kas tika apspriestas darbā ļoti atšķiras viena no otras un tas var būt izmantotas, lai risinātu vairākas atšķirīgas problēmas, kas saistītas ar virtuālo ainu renderēšanu. Darba apskatītie temati nav vienīgie, eksistē vel daudzas citas metodes, darba tika aprakstītas tikai pašas izplatītākas metodes.

Praktiskajā daļā ir aprakstīts, kā iegūt starpposma attēlu diviem kadriem, kas tika ņemt no viena skata punkta. Neviena no apskatītajam publikācijām nekas līdzīgs netika taisīts. Neskatoties uz to, ka rezultāta attēlu kvalitāte neatbilst sākuma attēlu kvalitātei, pēc būtības metode strādā efektīvi. Es uzskatu ka kvalitātes atšķirības starp ieejas attēliem un izejas attēlu ir realizācijas vaina, nevis metodes vaina, un var tikt izlabotas pielietojot citu attēlu metamorfēšanas metodi.

4 Secinājumi

Es uzskatu, ka man izdevās parādīt, ka izmantojot skatu metamorfēšanas metodi, var iegūt starpposma attēlu diviem ceļa satiksmes attēliem dažādos laika momentos. Protams eksistē arī citas metodes, kas varētu atrisināt šo problēmu.

Iespējami temati turpmākiem pētījumiem varētu būtu trifocal tenzora izmantošana šī darba problēmas risināšanai, kā arī pāreju iespējas no vienas plenoptiskas funkcijas parametrizācijas uz citu.

5 Pateicības

Gribētu izteikt pateicību sava darba vadītajam par organizatorisku palīdzību un metodiskiem ieteikumiem un saviem draugiem un paziņam par morālo atbalstu darba izstrādē.

6 Izmantotā literatūra un avoti

1. **Shum, H.Y., Chan, S.C., Kang, S.B.** *Image-Based Rendering.* Springer Science, 2007. 408 p.
2. **Shum, H.Y., Kang, S.B.** A Review of Image-based Rendering Techniques. *IEEE*, 2000, p. 2-13.
3. **Adelson, E. H., Bergen, J.** The plenoptic function and the elements of early vision. **In:** *Computational Models of Visual Processing.* MIT Press, Cambridge, MA, 1991, p. 3–20.
4. **McMillan, L., Bishop, G.** Plenoptic modeling: An image-based rendering system.**In:** *Computer Graphics (SIGGRAPH'95)*. 1995, p. 39–46.
5. **Levoy, M., Hanrahan, P.** Light field rendering. **In:** *Computer Graphics Proceedings, Annual Conference Series (SIGGRAPH'96)*, Proc. ACM SIGGRAPH, New Orleans, 1996, p. 31–42.
6. **Gortler, S. J., Grzeszczuk, R., Szeliski, R., Cohen, M. F.** The lumigraph. **In:** *Computer Graphics Proceedings, Annual Conference Series (SIGGRAPH'96)*, Proc. ACM SIGGRAPH, New Orleans, 1996, p. 43–54.
7. **Seitz, S. M., Dyer, C. M.** View morphing. **In:** *Computer Graphics Proceedings, Annual Conference Series(SIGGRAPH'96)*, Proc. ACM SIGGRAPH, New Orleans, ACM SIGGRAPH, 1996, p. 21–30.
8. **Chen, S., Williams, L.** View interpolation for image synthesis. **In:** *Computer Graphics (SIGGRAPH'93)*, 1993, p. 279–288.
9. **Mark, W., McMillan, L., Bishop, G.** Post-rendering 3d warping. **In:** *Proc. Symposium on I3D Graphics*, 1997, p. 7–16.
10. **Shade, J., Gortler, S., He, L.W., Szeliski, R.** Layered depth images. **In:** *Computer Graphics (SIGGRAPH'98)*, Proc. ACM SIGGRAPH, Orlando, 1998, p. 231–242.
11. **Szeliski, R., Shum, H.Y.** Creating full view panoramic image mosaics and texture-mapped models. **In:** *Computer Graphics (SIGGRAPH'97)*, 1997, p. 251–258.
12. **Szeliski, R., Shum, H.Y.** Creating Full View Panoramic Image Mosaics and Environment Maps, **In:** *Computer Graphics (SIGGRAPH '97)*, 1997.

7 Pielikumi

Izstrādātas programmas kods:

```
#include <iostream>

using namespace std;

#include "cv.h"
#include "cvaux.h"
#include "highgui.h"

const int MAX_COUNT = 8;

int ptsR[MAX_COUNT][2] ={{74, 104},{123, 87},{164, 103},{163,
158},{116, 183},{80, 157},{111, 124},{111, 124}};//cube
int ptsL[MAX_COUNT][2] ={{133, 103},{178, 86},{223, 103},{219,
156},{178, 182},{135, 155},{178, 124},{202, 113}};//cube

CvPoint2D32f pointsL[MAX_COUNT] = {0,0,0,0,0,0,0,0};
CvPoint2D32f pointsR[MAX_COUNT] = {0,0,0,0,0,0,0,0};
CvPoint2D32f pointsM2[MAX_COUNT] = {0,0,0,0,0,0,0,0};


int countL = 0;
int countR = 0;
int countM = 0;
int add_ptL = 0;
int add_ptR = 0;

void onMouseEventL( int event, int x, int y, int flags, void* param)
{
	if( event == CV_EVENT_LBUTTONDOWN && countL < MAX_COUNT)
	{
		cout << "onMouseEventL" << endl;
		cout << x << " " << y << endl;

		IplImage* image = (IplImage*) param;
		CvPoint point = cvPoint(x,y);
		cvCircle(image, point, 3, cvScalar(30*countL,255,255 -
30*countL),-1, 8,0);
		pointsL[countL++] = cvPointTo32f(point);
		add_ptL = 1;
	}
}

void onMouseEventR( int event, int x, int y, int flags, void* param)
{
	if( event == CV_EVENT_LBUTTONDOWN && countR < MAX_COUNT)
	{
		cout << "onMouseEventR" << endl;
		cout << x << " " << y << endl;

		IplImage* image = (IplImage*) param;
		CvPoint point = cvPoint(x,y);
		cvCircle(image, point, 3, cvScalar(30*countR,255,255 -
30*countR),-1, 8,0);
		pointsR[countR++] = cvPointTo32f(point);
```

```
            add_ptR = 1;
      }
}

void autoPointFill(IplImage* imageL, IplImage* imageR, bool load =
false)
{
      int n,m;
      for (n=0;n<MAX_COUNT;n++)
      {
            CvPoint pointL = cvPoint(ptsL[n][0],ptsL[n][1]);
            cvCircle(imageL, pointL, 3, cvScalar(30*countL,255,255 -
30*countL),-1, 8,0);
            pointsL[countL++] = cvPointTo32f(pointL);

            if(load)
            {
                  CvPoint pointR = cvPoint(ptsR[n][0],ptsR[n][1]);
                  cvCircle(imageR, pointR, 3,
cvScalar(30*countR,255,255 - 30*countR),-1, 8,0);
                  pointsR[countR++] = cvPointTo32f(pointR);
            }
      }
}

int main()
{
    cout << "Hello world!" << endl;
    //init
    cvNamedWindow("Left (1)");
    cvNamedWindow("Right (2)");
    cvNamedWindow("Trans");
    cvNamedWindow("Morph");
    cvNamedWindow("OpticalFlow");

      IplImage* srcL = cvLoadImage("imgL.jpg");
      IplImage* srcR = cvLoadImage("imgR.jpg");

      IplImage* srcL_O = cvCloneImage(srcL);
      IplImage* srcR_O = cvCloneImage(srcR);

      IplImage* dst = cvCloneImage(srcL);
      IplImage* opf = cvCloneImage(srcL);

      CvSize image_size = cvGetSize(srcL);

      IplImage* srcL_1C = cvCreateImage(image_size, IPL_DEPTH_8U, 1);
      IplImage* srcR_1C = cvCreateImage(image_size, IPL_DEPTH_8U, 1);

      cvConvertImage(srcL, srcL_1C);
      cvConvertImage(srcR, srcR_1C);

      //get points
      cvSetMouseCallback("Left (1)", onMouseEventL, (void*) srcL_O);
      cvSetMouseCallback("Right (2)", onMouseEventR, (void*) srcR_O);

      while( 1 ){
```

```
            cvShowImage("Left (1)", srcL_O );
            cvShowImage("Right (2)", srcR_O );

            if(countL == MAX_COUNT && countR == MAX_COUNT)
                  break;

            int key = cvWaitKey( 15 );
            if( key == 27 || key == 13) //enter || escape to break
                  break;

            //a - fill with precalculated points
            if( key == 97)
            {
                  autoPointFill(srcL_O, srcR_O);

                  break;
            }

            //z - fill both with precalculated points
            if( key == 122)
            {
                  autoPointFill(srcL_O, srcR_O, true);
                  break;
            }

      }
      cvShowImage("Left (1)", srcL_O );
      cvShowImage("Right (2)", srcR_O );

      //optical flow
      int corner_count = MAX_COUNT;
      char features_found[ MAX_COUNT ];
      float feature_errors[ MAX_COUNT ];

      if(countR == 0)
      {
            cout << "Looking for features ------------"<<endl;
            IplImage* eig_image = cvCreateImage( image_size,
IPL_DEPTH_32F, 1 );
            IplImage* tmp_image = cvCreateImage( image_size,
IPL_DEPTH_32F, 1 );

            int win_size = 15;
            //*
            CvSize pyr_sz = cvSize( srcL->width+8, srcR->height/3 );

            IplImage* pyrA = cvCreateImage( pyr_sz, IPL_DEPTH_32F, 1 );
            IplImage* pyrB = cvCreateImage( pyr_sz, IPL_DEPTH_32F, 1 );

            cvCalcOpticalFlowPyrLK( srcL_1C, srcR_1C, pyrA, pyrB,
pointsL, pointsR, corner_count,
                  cvSize( win_size, win_size ), 5, features_found,
feature_errors,
                   cvTermCriteria( CV_TERMCRIT_ITER | CV_TERMCRIT_EPS,
20, 0.3 ), 0 );
      }
```

```
	///draw optical flow
	for(int i = 0; i < corner_count; i++)
	{
		CvPoint p0 = cvPoint( cvRound( pointsL[i].x ), cvRound(
pointsL[i].y ) );
		CvPoint p1 = cvPoint( cvRound( pointsR[i].x ), cvRound(
pointsR[i].y ) );
		cvLine( opf, p0, p1, CV_RGB(255,0,0), 2 );
		cvCircle(opf, p0, 1, cvScalar(30*i,255,255 - 30*i),-1,
8,0);
		cvCircle(opf, p1, 1, cvScalar(30*i,255,255 - 30*i),-1,
8,0);

		cvCircle(srcR_O, p1, 3, cvScalar(30*i,255,255 - 30*i),-1,
8,0);
	}


	///fundamental matrix
	int numPoints = corner_count;
	CvMat* points1		= cvCreateMat(2,numPoints,CV_32F);
	CvMat* points2		= cvCreateMat(2,numPoints,CV_32F);
	CvMat* status		= cvCreateMat(1,numPoints,CV_32F);
	CvMat* fundMatrix	= cvCreateMat(3,3,CV_32F);

	//convert points to matrix
	for(int i = 0; i < corner_count; i++)
	{
		cvmSet(points1, 0, i, pointsL[i].x);
		cvmSet(points1, 1, i, pointsL[i].y);

		cvmSet(points2, 0, i, pointsR[i].x);
		cvmSet(points2, 1, i, pointsR[i].y);

		cout << "onMouseEventR" << endl;
		cout << pointsR[i].x << " " << pointsR[i].y << endl;
	}

	int num =
cvFindFundamentalMat(points1,points2,fundMatrix,CV_FM_RANSAC);
	if( num == 1 )
	{
		cout << "Fundamental matrix was found\n";
		cout << "|" << cvmGet(fundMatrix,0,0)  << "|" <<
cvmGet(fundMatrix,0,1) << "|" << cvmGet(fundMatrix,0,2) << endl;
		cout << "|" << cvmGet(fundMatrix,1,0)  << "|" <<
cvmGet(fundMatrix,1,1) << "|" << cvmGet(fundMatrix,1,2) << endl;
		cout << "|" << cvmGet(fundMatrix,2,0)  << "|" <<
cvmGet(fundMatrix,2,1) << "|" << cvmGet(fundMatrix,2,2) << endl;
	}
	else
	{
		cout << "Fundamental matrix was NOT found\n";
	}
	//fundamental matrix
	///morph code
	IplImage* srcL_SL = cvCloneImage(srcL);
```

```
	IplImage* srcR_SL = cvCloneImage(srcR);

	static CvMatrix3 matrix;

	matrix.m[0][0] = cvmGet(fundMatrix,0,0);
	matrix.m[0][1] = cvmGet(fundMatrix,0,1);
	matrix.m[0][2] = cvmGet(fundMatrix,0,2);
	matrix.m[1][0] = cvmGet(fundMatrix,1,0);
	matrix.m[1][1] = cvmGet(fundMatrix,1,1);
	matrix.m[1][2] = cvmGet(fundMatrix,1,2);
	matrix.m[2][0] = cvmGet(fundMatrix,2,0);
	matrix.m[2][1] = cvmGet(fundMatrix,2,1);
	matrix.m[2][2] = cvmGet(fundMatrix,2,2);

	CvMatrix3* matScan = &matrix;
	int numScanlines = 10;

	cvMakeScanlines(matScan,image_size,0,0,0,0,&numScanlines);

	int* lengthEpilines1 = new int[numScanlines];
	int* lengthEpilines2 = new int[numScanlines];

	int* scanlines1 = new int[4*numScanlines];
	int* scanlines2 = new int[4*numScanlines];

	cvMakeScanlines(matScan,image_size,scanlines1,scanlines2,lengthEp
ilines1,lengthEpilines2,&numScanlines);

	uchar* preWarpData1 = new uchar[max(srcL->width,srcL->height)
		*numScanlines*3];
	uchar* preWarpData2 = new uchar[max(srcL->width,srcL->height)
		*numScanlines*3];

	cvPreWarpImage(numScanlines, srcL, preWarpData1, lengthEpilines1,
		scanlines1);
	cvPreWarpImage(numScanlines, srcR, preWarpData2, lengthEpilines2,
		scanlines2);

	int* numRuns1 = new int[numScanlines];
	int* numRuns2 = new int[numScanlines];

	int* runs1 = new int[srcL->width*numScanlines];
	int* runs2 = new int[srcL->width*numScanlines];

	int* runCorrelation1 = new int[max(srcL->width,srcL->height)
		*numScanlines*3];
	int* runCorrelation2 = new int[max(srcL->width,srcL->height)
		*numScanlines*3];

	cvFindRuns(numScanlines, preWarpData1, preWarpData2,
		lengthEpilines1, lengthEpilines2, runs1, runs2, numRuns1,
		numRuns2);

	int* scanlinesMorphedImage = new int[numScanlines*2*4];
	int* numScanlinesMorphedImage = new int[numScanlines*2*4];

	cvDynamicCorrespondMulti(numScanlines, runs1, numRuns1, runs2,
```

```
            numRuns2, runCorrelation1, runCorrelation2);

      uchar* tmpDataImageDst = new uchar[max(srcL->width,srcL-
            >height)*numScanlines*3];

      float alp = 0.5;

      int* scanlinesA = new int[numScanlines*2*4];
      int* lenghts = new int[numScanlines*2*4];

      cvMakeAlphaScanlines(scanlines1, scanlines2,
            scanlinesMorphedImage, numScanlinesMorphedImage
            ,numScanlines, alp);

      cout << "!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!" <<endl;//draw
scanlines
      for(int i = 0; i < numScanlines; i++)
      {
            if(i % 10 == 0)
            {
                  CvPoint p0, p1;
                  p0 = cvPoint( cvRound( scanlines1[i*4] ), cvRound(
                        scanlines1[i*4 + 1] ) );
                  p1 = cvPoint( cvRound( scanlines1[i*4 + 2] ),
                        cvRound( scanlines1[i*4 + 3] ) );
                  cvLine( srcL_SL, p0, p1, CV_RGB(255,0,0), 1 );

                  p0 = cvPoint( cvRound( scanlines2[i*4] ), cvRound(
                        scanlines2[i*4 + 1] ) );
                  p1 = cvPoint( cvRound( scanlines2[i*4 + 2] ),
                        cvRound( scanlines2[i*4 + 3] ) );
                  cvLine( srcR_SL, p0, p1, CV_RGB(255,0,0), 1 );
            }
      }
      cout << "!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!" <<endl;

      cvMorphEpilinesMulti(numScanlines, preWarpData1, lengthEpilines1,
            preWarpData2, lengthEpilines2, tmpDataImageDst,
            numScanlinesMorphedImage, alp, runs1, numRuns1, runs2,
            numRuns2, runCorrelation1, runCorrelation2);

      cout << "morph " << endl;

      IplImage* morphImage = cvCreateImage(cvSize(srcL->width,srcL-
            >height),8,3);

      cvPostWarpImage(numScanlines, tmpDataImageDst,
            numScanlinesMorphedImage, morphImage,
            scanlinesMorphedImage);

      //cvDeleteMoire(morphImage);

      cvShowImage("Morph",morphImage);

      cvShowImage("left",srcL_SL);
      cvShowImage("right",srcR_SL);
      //morph code
```

```
	///draw the epilines

	cout << "------------------------------epilines" <<endl;
	CvMat* epiLines1  = cvCreateMat(3,numPoints,CV_32F);
	cvComputeCorrespondEpilines(points2, 2, fundMatrix, epiLines1);

	CvMat* epiLines2  = cvCreateMat(3,numPoints,CV_32F);
	cvComputeCorrespondEpilines(points1, 1, fundMatrix, epiLines2);

	//draw epilines
	for(int i = 0; i < corner_count; i++)
	{
		double a, b, c;
		double el0x, el1x, el0y, el1y;
		CvPoint el0, el1;

		//image 1
		//a*x + b*y + c = 0
		a = cvmGet(epiLines1,0,i);
		b = cvmGet(epiLines1,1,i);
		c = cvmGet(epiLines1,2,i);

		el0x = 0;
		el1x = image_size.width;

		el0y = (-c - a*el0x)/b;;
		el1y = (-c - a*el1x)/b;

		el0 = cvPoint( cvRound( el0x ), cvRound( el0y ) );
		el1 = cvPoint( cvRound( el1x ), cvRound( el1y ) );

		cout << "pointL" << el0.x << " | " << el0.y << " | " <<
			el1.x << " | " << el1.y << endl;

		cvLine( srcL_O, el0, el1, CV_RGB(255,0,0), 1 );

		//image 2
		//a*x + b*y + c = 0
		a = cvmGet(epiLines2,0,i);
		b = cvmGet(epiLines2,1,i);
		c = cvmGet(epiLines2,2,i);

		el0x = 0;
		el1x = image_size.width;

		el0y = (-c - a*el0x)/b;;
		el1y = (-c - a*el1x)/b;

		el0 = cvPoint( cvRound( el0x ), cvRound( el0y ) );
		el1 = cvPoint( cvRound( el1x ), cvRound( el1y ) );

		cout << "pointR" << el0.x << " | " << el0.y << " | " <<
			el1.x << " | " << el1.y << endl;

		cvLine( srcR_O, el0, el1, CV_RGB(255,0,0), 1 );
	}
```

```
	///calculate epiline interpolation
	CvMat* pointsM = cvCreateMat(2,numPoints,CV_32F);
	for(int i = 0; i < corner_count; i++)
	{
		double pMx, pMy;
		double s = 0.5;
		pMx = (1-s) * cvmGet(points1, 0, i) + s * cvmGet(points2,
			0, i);
		pMy = (1-s) * cvmGet(points1, 1, i) + s * cvmGet(points2,
			1, i);

		cvmSet(pointsM, 0, i, pMx);
		cvmSet(pointsM, 1, i, pMy);

		CvPoint point = cvPoint(pMx,pMy);
		pointsM2[countM++] = cvPointTo32f(point);
	}

	///homography
	CvMat* homMatrix1M	= cvCreateMat(3,3,CV_32F);
	CvMat* homMatrix2M	= cvCreateMat(3,3,CV_32F);

	cvFindHomography(points1, pointsM, homMatrix1M, CV_RANSAC, 4);
	cvFindHomography(points2, pointsM, homMatrix2M, CV_RANSAC, 4);

	IplImage* hom1M = cvCloneImage(srcL);
	IplImage* hom2M = cvCloneImage(srcL);

	cvWarpPerspective(srcL, hom1M, homMatrix1M);
	cvWarpPerspective(srcR, hom2M, homMatrix2M);

	///overlap
	double alpha = 0.5;
	double beta = 0.5;

	cvSetImageROI(dst, cvRect(0,0,image_size.width,
		image_size.height));
	cvAddWeighted(hom1M, alpha, hom2M, beta, 0.0, dst);
	cvResetImageROI(dst);
	//overlap

	//*
	for(int i = 0; i < corner_count; i++)
	{
		CvPoint pM = cvPoint( cvRound( cvmGet(pointsM, 0, i) ),
			cvRound( cvmGet(pointsM, 1, i) ) );

		cvCircle(dst, pM, 3, cvScalar(0,255,0),-1, 8,0);

		cvCircle(hom1M, pM, 3, cvScalar(30*i,255,255 - 30*i),-1,
			8,0);
		cvCircle(hom2M, pM, 3, cvScalar(30*i,255,255 - 30*i),-1,
			8,0);

		CvPoint p1 = cvPoint( cvRound( cvmGet(points1, 0, i) ),
			cvRound( cvmGet(points1, 1, i) ) );
		CvPoint p2 = cvPoint( cvRound( cvmGet(points2, 0, i) ),
```

```
                    cvRound( cvmGet(points2, 1, i) ) );
        }

        cvShowImage("hom1M", hom1M);
        cvShowImage("hom2M", hom2M);

        //show
        cvShowImage("Left (1)", srcL_O);
        cvShowImage("Right (2)", srcR_O);
        cvShowImage("OpticalFlow", opf);
        cvShowImage("Trans", dst);

        //wait
        cvWaitKey(0);

        //clean up
        cvReleaseImage(&opf);
        cvReleaseImage(&dst);
        cvReleaseImage(&srcL);
        cvReleaseImage(&srcR);

        cvDestroyAllWindows();
    return 0;
}
```

Bakalaura darbs

UZ ATTĒLIEM BALSTĪTA RENDERĒŠANA

Ar savu parakstu apliecinu, ka pētījums veikts patstāvīgi, izmantoti tikai tajā norādītie informācijas avoti un iesniegtā darba elektroniskā kopija atbilst izdrukai. Piekrītu sava darba publicēšanai internetā.

Autors: _____
(Autora paraksts)

Ar savu parakstu apliecinu, ka esmu lasījis augšminēto bakalaura darbu un atzīstu to par **piemērotu/nepiemērotu** (nevajadzīgo svītrot) aizstāvēšanai Latvijas Universitātes datorzinātņu bakalaura studiju programmas gala pārbaudījuma komisijas sēdē.

Darba vadītājs(-ja): _____
(Vadītāja paraksts)

Darbs iesniegts Datorikas fakultātē _____.
(Iesniegšanas datums)

Ar šo es apliecinu, ka darba elektroniskā versija ir augšupielādēta LU informatīvajā sistēmā.

Metodiķe: _____.
(Metodiķes paraksts)

Recenzents: _____

Darbs aizstāvēts bakalaura darbu gala pārbaudījuma komisijas sēdē

_____ prot. Nr. _____, vērtējums _____
(Darba aizstāvēšanas datums)

Komisijas sekretārs: _____
(Sekretāra paraksts)